



AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

**WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI,
INFORMATYKI I INŻYNIERII BIOMEDYCZNEJ**

KATEDRA INFORMATYKI STOSOWANEJ

Praca dyplomowa inżynierska

*Implementacja systemu uwierzytelniania z zastosowaniem
Negatywnych Baz Danych*

Implementation of authentication system using Negative Databases

Autor:

Grzegorz Nieużyła

Kierunek studiów:

Informatyka

Opiekun pracy:

dr inż. Piotr Szwed

Kraków, 2020

Uprzedzony o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.): „Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystycznego wykonania albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, wideogram lub nadanie.”, a także uprzedzony o odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (t.j. Dz. U. z 2012 r. poz. 572, z późn. zm.): „Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godności studenta student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskim samorządu studenckiego, zwanym dalej «sądem koleżeńskim».”, oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

Spis treści

1. Wprowadzenie	7
1.1. Cele pracy	7
1.2. Zawartość pracy	8
2. Solwery SAT	9
2.1. Problem spełnialności	9
2.2. Opis działania solwerów SAT	10
2.2.1. Solwery kompletne	10
2.2.2. Solwery niekompletne	12
3. Negatywne Bazy Danych - opis teoretyczny	13
3.1. Opis działania	13
3.2. Zastosowanie w systemach uwierzytelniania	14
3.3. Algorytmy generacji Negatywnych Baz Danych	14
3.3.1. Algorytm prefiksowy	14
3.3.2. Algorytm <i>Randomize_NDB</i>	16
3.3.3. Algorytm <i>0-Hidden</i>	20
3.3.4. Algorytmy <i>1-Hidden</i> i <i>2-Hidden</i>	20
3.3.5. Algorytm <i>Q-Hidden</i>	21
3.3.6. Algorytm <i>K-Hidden</i>	23
3.3.7. Algorytm hybrydowy	24
3.4. Wykorzystywanie solwerów SAT w celu uzyskania przeciwobrazu Negatywnej Bazy Danych	26
4. Testy algorytmów	29
4.1. Opis testowania za pomocą solwerów SAT	29
4.2. Rozkład statystyczny <i>NDB</i>	30
4.3. Testy metod weryfikujących poprawność rekordów pozytywnych	31
4.3.1. Wpływ parametrów generacji na ilość niepożądanych ukrytych ciągów bitowych	32
4.3.2. Skuteczność funkcji sum kontrolnych	33

4.4.	Testy algorytmów generacji <i>NDB</i>	34
4.4.1.	Algorytm prefiksowy	34
4.4.2.	Algorytm <i>Randomize_NDB</i>	34
4.4.3.	Algorytm <i>Q-Hidden</i>	36
4.4.4.	Algorytm <i>K-Hidden</i>	41
4.4.5.	Algorytm hybrydowy	42
5.	Implementacja systemu uwierzytelniania	45
5.1.	Reprezentacja danych	45
5.2.	Działanie aplikacji	46
5.3.	Szczegóły implementacji	47
5.3.1.	Architektura systemu	47
5.3.2.	Algorytm tworzenia użytkownika	48
5.3.3.	Algorytm uwierzytelniania użytkownika	49
5.3.4.	Algorytm zmiany hasła	49
6.	Wnioski	51
6.1.	Bezpieczeństwo przedstawionej implementacji	51
6.2.	Możliwe rozszerzenia	51

1. Wprowadzenie

1.1. Cele pracy

Celem niniejszej pracy jest implementacja i przetestowanie systemu uwierzytelniania oferującego większe bezpieczeństwo niż standardowy schemat generowania skrótu hasła za pomocą funkcji generacji klucza (np. *PBKDF2*, *bcrypt*) i przechowywaniu w standardowej (pozytywnej) bazie danych.

Jak wynika z raportu Google[1], większość internautów ma problem z zachowaniem zasad bezpieczeństwa dotyczącego używania mediów społecznościowych i portali internetowych. 66% procent badanych przyznało się do używania jednego hasła do różnych kont. W przypadku wykradzenia bazy danych pojedynczego portalu i uzyskaniu postaci jawnych haseł istnieje ryzyko uzyskania dostępu również do innych aplikacji przez atakującego. W takiej sytuacji wyciek danych z amatorskiego forum tematycznego może zagrozić naszemu kontu bankowemu lub poczcie elektronicznej.

Główną przyczyną nadużywania jednego hasła jest konieczność zapamiętania danych logowaniu do wielu stron internetowych – 75% internautów ma problem z ich zarządzaniem.

Istnieje szereg metod, które indywidualny użytkownik może zastosować w celu polepszenia swojego bezpieczeństwa w Internecie, ale nie cieszą się one dużą popularnością. Według raportu tylko 37% internautów używa uwierzytelnienia wielopoziomowego, a menadżerów haseł umożliwiających stosowanie różnych i silnych danych logowania – 15%.

Częściowym rozwiązaniem tego problemu może być wprowadzenie dodatkowych warstw bezpieczeństwa bezpośrednio w systemach uwierzytelniania w celu utrudnienia uzyskania nieautoryzowanego dostępu do wrażliwych danych. Takie podejście pozwoli podnieść bezpieczeństwo wszystkich użytkowników, niezależnie od indywidualnych preferencji i nawyków.

Założeniem systemu jest zamienienie reprezentacji w sposób jawny na Negatywną Bazę Danych (*NDB*) co pozwoli dodać dodatkową warstwę bezpieczeństwa która znacząco utrudni uzyskanie haseł użytkownika w przypadku kradzieży bazy danych.

W tym celu opisałem różne algorytmy prezentowane w dostępnej literaturze, przedstawiłem schemat ich działania i porównałem je pod względem bezpieczeństwa.

Rezultatem wyjściowym algorytmów generacji *NDB* jest zbiór ciągów tekstowych, które można jednoznacznie sprowadzić do zbioru formuł logicznych CNF, dlatego przeprowadzone zostały testy z wykorzystaniem solverów SAT mające na celu zasymulowanie ataku na powstałą *NDB*.

Koncept negatywnej reprezentacji danych wywodzi się z immunologii. Badania wykazały, że limfocyty T wykorzystują podobny proces w celu rozróżnienia między sobą i innymi komórkami [2].

1.2. Zawartość pracy

Niniejsza praca jest podzielna na sześć rozdziałów. W rozdziale 1 przedstawiam obecne problemy występujące w systemach uwierzytelniania i wstępnie opisuję schemat działania Negatywnych Baz Danych.

W rozdziale 2 opisuję problem spełnialności oraz metody i techniki stosowane przez współczesne solwery SAT.

Rozdział 3 poświęcony jest dostępnym w literaturze algorytmom generacji Negatywnych Baz Danych. Porównywane są pod kątem teoretycznym na podstawie przytoczonych artykułów.

Dokładne testy metod generacji *NDB* są przedstawione w rozdziale 4. Opisany tam jest wpływ wszystkich dostępnym parametrów na trudność ich odwrócenia za pomocą solwerów SAT zChaff oraz WalkSAT. Dodatkowo porównywane są metody zapobiegania występowaniu niepożądanych rekordów pozytywnych w generowanej *NDB*.

W rozdziale 5 opisuję stworzoną na potrzeby tej pracy implementację systemu uwierzytelniania wykorzystującą Negatywne Bazy Danych.

Wnioski wyniesione z analizowania konceptu *NDB* i pisanie tej pracy znajdują się w rozdziale 6.

2. Solwery SAT

2.1. Problem spełnialności

Problem spełnialności (ang. *Boolean satisfiability problem*, w skrócie SAT) polega na sprawdzeniu, czy dana formuła logiczna posiada odpowiednie przypisanie zmiennych przy których całość wyrażenia będzie ewaluowana do prawdy.

Wyrażenia wykorzystywane w problemie SAT składają się z:

- zmiennych - $x_1, x_2 \dots x_n$, mogących przybrać wartość prawdziwą (*true*, T , \top , 1) lub fałszywą (*false*, F , \perp , 0)
- operatora koniunkcji - *AND*, \wedge
- operatora alternatywy - *OR*, \vee
- operatora negacji - *NOT*, \neg
- nawiasów - $()$

Formuły SAT przedstawia się w postaci CNF (koniunkcyjna postać normalna, ang. *conjunctive normal form*) której struktura jest następująca: formuła jest koniunkcją klauzul, a klauzula jest alternatywą literałów (tj. zmiennych lub ich negacji):

$$(x_{11} \vee x_{12} \vee \dots \vee x_{1n}) \wedge (x_{21} \vee x_{22} \vee \dots \vee x_{2n}) \wedge \dots \wedge (x_{m1} \vee x_{m2} \vee \dots \vee x_{mn})$$

Naiwny algorytm SAT jest trywialny - wystarczy dla każdej możliwej kombinacji zmiennych $x_1 \dots x_n$ sprawdzić czy formuła jest spełnialna. Sprawdzenie następuje w czasie liniowym od wartości n , zatem cały algorytm ma złożoność obliczeniową $O(2^n)$ - nie do przyjęcia w praktycznie żadnych zastosowaniach poza bardzo małymi wartościami n .

Problem SAT jest pierwszym problemem dla którego udowodniono że jest *NP*-zupełny (Twierdzenie Cooka-Levina [3, 4]). Oznacza to, że można w czasie wielomianowych sprowadzić każdy problem decyzyjny zawierający się w *NP* do SAT.

Problem należący do *NP* charakteryzuje się tym, że zweryfikowanie pojedynczego rozwiązania jest możliwe w czasie wielomianowym dla deterministycznej Maszyny Turinga, oraz że sprawdzenie wszystkich możliwości (a więc rozstrzygnięcie problemu) jest możliwe również w czasie wielomianowym, ale

dla niedeterministycznej Maszyny Turinga - przez jednoczesne sprawdzenie każdej kombinacji. Wynika z tego również, że $P \subseteq NP$.

Nie każda możliwa formuła CNF stanowi problem NP -zupełny. Twierdzenie o dychotomii Schaefera określa szczególne instancje problemu należące do P (przy założeniu, że $P \neq NP$) [5]:

1. Formuła jest spełnialna jeśli wszystkie zmienne przyjmują wartość 0
2. Formuła jest spełnialna jeśli wszystkie zmienne przyjmują wartość 1
3. Każda klauzula ma co najwyżej jeden literal pozytywny (każda klauzula jest klauzulą Horna)
4. Każda klauzula ma co najwyżej jeden literal negatywny (każda klauzula jest dualną klauzulą Horna)
5. Każda klauzula ma co najwyżej dwa literały (2-CNF)
6. Formuła jest równoznaczna systemowi równań linowych $x_1 \oplus x_2 \oplus \dots \oplus x_n = c$, gdzie $x_i, c \in \{0, 1\}$ a $\oplus \equiv$ dodawanie mod_2

2.2. Opis działania solwerów SAT

Solwery SAT można podzielić na dwie kategorie – kompletne i niekompletne. Algorytmy kompletne gwarantują, że jeśli istnieje spełniające przypisanie dla analizowanej formuły to zostanie zwrócony wynik pozytywny. Natomiast wynik działania solwera niekompletnego może być albo pozytywny – istnieje odpowiednie przypisanie, albo niezdefiniowany – algorytm nie znalazł rozwiązania w określonej liczbie prób.

2.2.1. Solwery kompletne

Algorytmy kompletne do rozstrzygania problemów SAT korzystają z szeregu metod do których należą: kwantyfikacja egzystencjalna, wnioskowanie, przeszukiwanie oraz wnioskowanie i przeszukiwanie [6].

2.2.1.1. Kwantyfikacja egzystencjalna

Kwantyfikacja egzystencjalna jest klasą algorytmów upraszczających zadaną formułę CNF do postaci spełnialnej $\{\}$ lub niespełnialnej $\{\emptyset\}$. W tym celu definiuje się operator \exists zmiennej P w formule Δ :

$$\exists P \Delta \stackrel{\text{def}}{=} (\Delta | P) \vee (\Delta | \neg P)$$

gdzie operator $|$ jest *warunkowaniem* i definiuje się je następująco:

$$\Delta | P = \{\alpha - \{\neg P\} \mid \alpha \in \Delta, P \notin \alpha\} \quad [6]$$

Proces wnioskowania można intuicyjnie przedstawić jako ustawienie zmiennej P na wartość prawdziwą (lub fałszywą w przypadku $\neg P$) i zredukowanie zbioru klauzul usuwając te, które są spełnione oraz usuwając z pozostałych literały mające wartość fałszywą. Kwantyfikator $\exists P \Delta$ określa formułę po wyeliminowaniu zmiennej P przez rozpatrzenie obu możliwych wartości.

Z punktu widzenia problemu spełnialności najważniejszą właściwością jest to, że formuła Δ jest spełnialna wtedy i tylko wtedy, gdy $\exists P \Delta$ jest spełnialna. Pozwala to na stopniowe eliminowanie zmiennych aż do osiągnięcia pustej klauzuli lub pustej formuły. Kwantyfikacja egzystencjalna jest wykorzystywana m.in. w algorytmie DP oraz w symbolicznym rozwiązywaniu SAT.

2.2.1.2. Wnioskowanie

Upraszczenie formuł przez wnioskowanie opiera się na usuwaniu redundantnych literałów tj. wykluczających się nawzajem. Przykładowo w formule:

$$\{\{x_1, x_2, \neg x_3\}, \{\neg x_2, x_4, x_5\}\}$$

zmienna x_2 występuje w postaci zanegowanej i niezaniegowanej w różnych klauzulach. Oznacza to, żeby cała formuła była spełniona, co najmniej jeden pozostały literał musi się ewaluować do prawdy. Można zatem zastosować następujące przekształcenie:

$$\{\{x_1, \neg x_3, x_4, x_5\}\}$$

Dana formuła jest niespełnialna wtedy i tylko wtedy, gdy istnieje wnioskowanie prowadzące do pustej klauzuli, zatem po wyczerpaniu wszystkich możliwości wnioskowania możemy ustalić, że problem jest spełnialny.

Do algorytmów wykorzystujących tę metodę należą algorytm Stålmarcka i HeerHugo.

2.2.1.3. Przeszukiwanie

Problem SAT można przedstawić jako drzewo decyzyjne przypisania zmiennych, gdzie liście są wszystkimi możliwymi kombinacjami literałów. Metoda przeszukiwania polega na znalezieniu liścia spełniającego formułę. Używa się do tego algorytmu przeszukiwania wszerz oraz warunkowania w celu ograniczenia drzewa.

Dodatkowo używa się także *propagacji jednostkowej* (ang. *unit propagation*) za pomocą której można odrzucić lub potwierdzić poprawność danego poddrzewa we wczesnej fazie analizy przez spełnienie klauzul jednostkowych (posiadających tylko jeden literał).

Wzorcowym zastosowaniem tej metody jest algorytm *DPLL*.

2.2.1.4. Wnioskowanie i przeszukiwanie

Wadą podejścia zastosowanego w *DPLL* jest fakt, że podczas cofania się po drzewie w kierunku korzenia nie bierze się pod uwagę zbioru zmiennych, które spowodowały odrzucenie danego poddrzewa. Dlatego w tej metodzie stosuje się tzw. niechronologiczne cofanie – podczas konfliktu literałów wyznacza się zbiór powodujących go zmiennych i zmienia się ich wartości, pomijając te nie biorące udziału w konflikcie.

Dodatkowo podczas działania algorytmu wprowadza się dodatkowe klauzule przedstawiające zależności, które pozwalają wykrywać nieścisłości bliżej korzenia, jeszcze bardziej skracając czas działania.

Do tej klasy należy większość współczesnych, najbardziej optymalnych algorytmów kompletnych rozstrzygania problemu SAT – m.in. *zChaff*, *MiniSAT*, *PicoSAT* i *Siege*.

2.2.2. Solwery niekompletne

Solwery nie gwarantujące znalezienia rozwiązania opierają się na *stochastycznym przeszukiwaniu lokalnym*. Takie rozwiązanie pozwala na znaczne mniejsze czasy wykonywania dla odpowiednich formuł niż algorytmy wykorzystujące DPLL, ale w przypadku trudnych problemów zamiast zdegenerowania złożoności obliczeniowej do $O(2^n)$ zostaje zwrócony wynik nieokreślony.

Czołowymi solwerami z tej kategorii są *GSAT* i *WalkSAT* [7].

2.2.2.1. GSAT

Schemat działania *GSAT* polega na wygenerowaniu losowego przypisania i zachłannym *odwracaniu* (ang. *flipping*) wartości zmiennych w celu zminimalizowania liczby niespełnionych klauzul. Ilość odwróceń oraz liczbę prób z wygenerowaniem nowego ciągu początkowego można modyfikować za pomocą odpowiednich parametrów.

2.2.2.2. WalkSAT

WalkSAT rozszerza algorytm stopniowego odwracania zmiennych używany w *GSAT*, redukując stopień zachłanności. Na początku każdej iteracji wybierana jest losowa klauzula, i w niej szukana jest zmienna, której zanegowanie nie zmieni żadnej innej klauzuli na niespełnioną. Jeśli nie została znaleziona, *WalkSAT* wybiera z danej klauzuli losową zmienną lub taką, której zmiana spowoduje spełnienie jak największej ilości pozostałych klauzul. Wprowadzony jest dodatkowo parametr szumu $p \in [0, 1]$ określający jak często wybierane jest podejście nie-zachłanne. Dla losowych formuł 3SAT optymalna wartość szumu wynosi $p = 0.57$ [7].

3. Negatywne Bazy Danych - opis teoretyczny

3.1. Opis działania

Główną operacją wykonywalną na *NDB* jest sprawdzenie czy dany rekord znajduje się w bazie. Przyjmując U jako oznaczenie uniwersum języka binarnego o długości l a DB jako zbiór wszystkich rekordów, każdy o długości l , *NDB* przechowuje zbiór $U - DB$ [8]. Takie dane są niepraktycznie do zareprezentowania w postaci nieskompresowanej z uwagi na wielkość, dlatego stosuje się wyrazy nad alfabetem $\{0, 1, *\}$ gdzie symbol $*$ może oznaczać zarówno 0 lub 1 w jawnej reprezentacji bitowej. Pozycje na których znajduje się wartość 0 lub 1 są *ustalone* a z wartością $*$ - *nieustalone*.

Każdy taki wyraz odpowiada jednemu lub wielu elementom $U - DB$ i jest sprowadzany do formuły logicznej (Tabela 3.1). Z założenia algorytm sprawdzający przynależność do DB sprawdza czy jakakolwiek formuła z *NDB* jest spełniana przez dany rekord. Dane znajdują się w DB wtedy i tylko wtedy gdy żadna formuła nie zostanie spełniona.

Taki model działania wymusza na danych stałą wielkość, co jednak nie stanowi problemu w przypadku przechowywania skrótów haseł które mają stałą, zależną od konkretnego algorytmu długość. Dla danych o zmiennych rozmiarach (np. nazwy użytkownika) można zastosować funkcję hashującą lub algorytmy zwiększające długość ciągu bitowego do stałej wartości typu PKCS#5 lub PKCS#7. Należy jednak pamiętać, że zwiększenie długości rekordu znacznie wydłuża czas generacji bazy oraz zajmowaną pamięć dla niektórych algorytmów.

rekord NDB	formuła logiczna
011*	$\neg x_1 \wedge x_2 \wedge x_3$
001*	$\neg x_1 \wedge \neg x_2 \wedge x_3$
1*1*	$x_1 \wedge x_3$
0*0*	$\neg x_1 \wedge \neg x_3$
1*00	$x_1 \wedge \neg x_3 \wedge \neg x_4$

Tabela 3.1. Reprezentacja formuł logicznych za pomocą NDB kodującej zbiór $DB = \{1001, 1101, 1110\}$

3.2. Zastosowanie w systemach uwierzytelniania

NDB może być wykorzystana w każdym systemie, gdzie podstawową operacją na danych jest sprawdzenie czy dany rekord znajduje się w bazie. Jednym z najpopularniejszych systemów uwierzytelniania jest metoda oparta na loginie i hasle. Użytkownik danej aplikacji przy zakładaniu konta podaje hasło, które następnie warstwa serwerowa danej aplikacji przechowuje jako wynik nieodwracalnej funkcji hashującej.

W przypadku nieautoryzowanego dostępu do bazy danych i używanego algorytmu uzyskiwania skrótu z hasła, atakujący może uzyskać wartość pierwotną mało skomplikowanych haseł za pomocą np. metody słownikowej. Modyfikując powyższy algorytm składując skróty jako rekordy w NDB uniemożliwiamy łatwą iterację wszystkich danych, jednocześnie pozostawiając łatwy dostęp do informacji czy użytkownik o podanym loginie i hasle ma dostęp do aplikacji.

3.3. Algorytmy generacji Negatywnych Baz Danych

3.3.1. Algorytm prefiksowy

Najprostszym ze sposobów generowania Negatywnych Baz Danych jest zaproponowany przez Fernando Esponda algorytm prefiksowy [8, 9]. Został on opracowany w celu udowodnienia że proces generowania NDB z rekordów DB jest możliwy w rozsądnej złożoności czasowej i pamięciowej.

Algorytm 1: Algorytm prefiksowy

Dane: DB - zbiór rekordów do zarepresentowania w NDB, l - długość rekordu DB

Wynik: Zbiór rekordów NDB

```

1  $Prefix_n(V)$  - Prefiks  $n$ -znakowy rekordu  $V$ 
2  $len(V)$  - Długość rekordu  $V$ 
3  $W_i = \{ \}$ ;
4  $i = 0$ ;
5 while  $i < l$  do
6    $W_{i+1} =$  Zbiór wszystkich  $i + 1$ -znakowych ciągów bitowych  $V_p$  nie będących prefiksem
      żadnego rekordu  $DB$  i dla których  $Prefix_i(V_p) \in W_i$ 
7   foreach  $V_p$  in  $W_{i+1}$  do
8     Stwórz rekord NDB o długości  $l$  którego  $V_p$  jest prefiksem a na pozostałych pozycjach
      jest symbol  $*$  i dodaj do zbioru wyjściowego NDB
9   end
10   $i = i + 1$ ;
11   $W_i =$  Zbiór wszystkich  $i$ -znakowych prefiksów rekordów  $DB$ 
12 end
```

Powyższa metoda polega na generowaniu coraz dłuższych prefiksów które nie pokrywają się ze zbiorem DB . W ten sposób na początku tworzone są rekordy odpowiadające znacznej części $U - DB$. Czasami występuje potrzeba zdefiniowania pewnych rekordów explicite bez wykorzystania symbolu $*$ jeżeli każdy możliwy prefiks jest także prefiksem rekordu z DB . Przykładowy wynik działania znajduje się w tabeli 3.2.

DB	U - DB	NDB
0000	0001	10**
0110	0011	010*
0010	0100	111*
1101	0101	0001
	0111	0011
	1000	0111
	1001	1100
	1010	
	1011	
	1100	
	1110	
	1111	

Tabela 3.2. Rezultat działania algorytmu prefiksowego

Algorytm prefiksowy jest deterministyczny i każdy powstały rekord reprezentuje unikalną, nie pokrywającą się część $U - DB$ [8]. Powoduje to, że algorytm uzyskiwania zbioru DB z otrzymanej NDB

nie wymaga sprowadzenia do problemu SAT. Wystarczy jedynie odpowiednio posortować rekordy i wyznaczyć przedziały pomiędzy nimi, co pokazują w algorytmie 2.

Algorytm 2: Algorytm odwracający *NDB* wygenerowaną za pomocą algorytmu prefiksowego

Dane: *NDB* - zbiór rekordów do odwrócenia, l - długość rekordu *NDB*

Wynik: Zbiór rekordów DB

- 1 Posortuj zbiór *NDB*
 - 2 V_l = rekord *DB* wypełniony symbolami '0' o długości l
 - 3 **foreach** V **in** *NDB* **do**
 - 4 $V_h = V$ z zamienionymi symbolami '*' na '0'
 - 5 $R = \text{Rekordy_DB_W_Przedziale}(V_l, V_h)$
 - 6 Dodaj R do zbioru wynikowego
 - 7 $V_l = V$ z zamienionymi symbolami '*' na '1'
 - 8 **end**
 - 9 V_h = rekord *DB* wypełniony symbolami '1' o długości l
 - 10 $R = \text{Rekordy_DB_W_Przedziale}(V_l, V_h)$
 - 11 Dodaj R do zbioru wynikowego
-

Algorytm 3: *Rekordy_DB_W_Przedziale*

Dane: V_1, V_2 - rekordy *DB*

Wynik: Zbiór rekordów DB w przedziale (V_1, V_2)

- 1 Dodaj binarnie 1 do V_1
 - 2 **while** $V_1 \neq V_2$ **do**
 - 3 Dodaj V_1 do zbioru wynikowego.
 - 4 Dodaj binarnie 1 do V_1
 - 5 **end**
-

Czas wykonywania procedury generacyjnej wynosi $O(l|DB|)$, jednak w przypadku zapisywania wyniku do bazy wzrasta do $O(l^2|DB|)$ gdyż konieczna jest serializacja każdego rekordu. Złożoność pamięciowa dla optymalnej implementacji wynosi $O(l|DB|)$ w przypadku gdy poprzednio generowane rekordy *NDB* nie są przetrzymywane w pamięci. Dla danego zbioru *DB* generowane jest $O(l|DB|)$ rekordów co sprowadza się do wielkości powstałej bazy danych wynoszącej $O(l^2|DB|)$.

Przestawiony powyżej algorytm odwracający (2) wykonuje się w czasie $O(|NDB|) = O(l|DB|)$ co jest wynikiem lepszym od procedury generacyjnej z zapisem do bazy.

3.3.2. Algorytm *Randomize_NDB*

Algorytm prefiksowy generuje poprawne rekordy *NDB*, jednak nie jest praktyczny w żadnych zastosowaniach związanych z bezpieczeństwem, ponieważ wynik jego działania jest stosunkowo prosto

sprowadzić do postaci pozytywnej. Aby temu zaradzić, Fernando Esponda w swojej pracy[8] zaproponował niedeterministyczny algorytm mający na celu wprowadzić rekordy które nie odpowiadają jedynie prefiksom elementów z $U - DB$ i są trudniejsze do odwrócenia.

Algorytm 4: Algorytm *Randomize_NDB*

Dane: DB - zbiór rekordów do zarepresentowania w NDB, l - długość rekordu DB

Wynik: Zbiór rekordów NDB

```

1   $Prefix_n(V)$  - Prefiks  $n$ -znakowy rekordu  $V$ 
2   $len(V)$  - Długość rekordu  $V$ 
3   $\pi$  = losowa permutacja o długości  $|V_{pe}|$ 
4   $W_i$  = zbiór wszystkich ciągów  $l$ -bitowych
5   $\pi(DB) \equiv \{\pi(V) \mid V \in DB\}$ 
6   $i = \lceil \log_2(l) \rceil$ ;
7  while  $i < l$  and  $W_i \neq \emptyset$  do
8       $W_{i+1}$  = Zbiór wszystkich  $i + 1$ -znakowych ciągów bitowych  $V_p$  nie będących prefiksem
          żadnego rekordu  $\pi(DB)$  i dla których  $Prefix_i(V_p) \in W_i$ 
9      foreach  $V_p$  in  $W_{i+1}$  do
10         Dopełnij  $V_p$  do długości  $l$  wstawiając znaki nieustalone '*' na końcu
11          $j$  = losowa liczba z przedziału  $[1, l]$ 
12         for  $k = 1$  to  $j$  do
13              $n$  = losowa liczba z przedziału  $[1, \log_2(l)]$ 
14              $P = n$  losowych nieustalonych pozycji z  $V_p$ 
15              $X$  = Zbiór rekordów powstałych przez zastąpienie pozycji  $\in P$  przez wszystkie
                 możliwe kombinacje bitowe ( $2^n$  rekordów)
16             foreach  $V_q$  in  $X$  do
17                  $V_{pg} = \text{Pattern\_Generate}(\pi(DB), V_q)$ 
18                 Dodaj  $\pi^{-1}(V_{pg})$  do zbioru rekordów wyjściowych
19             end
20         end
21     end
22      $i = i + 1$ 
23      $W_i$  = Zbiór wszystkich  $i$ -znakowych prefiksów rekordów  $DB$ 
24 end

```

Algorytm ten działa na podobnej zasadzie co algorytm prefiksowy (rozdział 3.3.1) z pewnymi modyfikacjami. Na początku kolejność bitów w DB jest mieszana za pomocą losowej permutacji aby pozycje zdefiniowane w generowanej prefiksowej NDB nie były skumulowane na początku wyrazów. Następnie dla każdego powstałego negatywnego rekordu losuje się n pozycji nieustalonych i zastępuje

się go równoznacznym zbiorem rekordów które mają te pozycje ustalone. Powstałe ciągi są dodatkowo zaciemniane przez wstawienie na losowych pozycjach zamiast bitu zdefiniowanego znak $*$ zgodnie z algorytmem *Pattern_Generate*.

Wynik algorytmu jest niedeterministyczny co powoduje że dla takich samych zbiorów *DB* rezultat może się różnić. Generacja wielu redundantnych rekordów zwiększa odporność otrzymanej bazy na próby przywrócenia do postaci pozytywnej, jednak wiąże się to z ze zwiększeniem objętości NDB średnio $\frac{l^2}{2}$ razy w stosunku do algorytmu prefikсового.

Algorytm 5: Algorytm *Pattern_Generate*

Dane: *DB* - zbiór rekordów do zarepresentowania w NDB, V_q - rekord NDB do zaciemnienia

Wynik: Zaciemniony rekord NDB

```

1   $\pi$  = losowa permutacja o długości  $|V_{pe}|$ 
2   $SIV = \{ \}$  // wektor zmienionych bitów
3  for  $i = 1$  to  $|V_q|$  do
4       $\pi(V_q)' = \pi(V_q)$  z elementem o indeksie  $i$  zamienionym na symbol  $*$ 
5      if istnieje rekord w DB pokrywający się z  $\pi(V_q)'$  then
6          Dodaj  $i$  i bit o indeksie  $i$  do  $SIV$ 
7           $\pi(V_q) = \pi(V_q)'$ 
8      end
9  end
10  $t$  = losowa liczba z przedziału  $[0, |SIV|]$ 
11 if  $t > |SIV|$  then
12      $R = SIV$ 
13 else
14      $R = t$  losowych bitów z  $SIV$ 
15 end
16  $V_k = \pi(V_q)$ 
17 for indeks, bit in  $R$  do
18      $V_k[indeks] = bit$ 
19 end
20 Zwróć  $\pi^{-1}(V_k)$ 

```

Zmiany względem algorytmu prefiksowego oraz procedura *Pattern_Generate* gwarantują że żaden ze zmodyfikowanych rekordów nie będzie odpowiadał żadnemu elementowi *DB* oraz że wynik działania pozostanie kompletnym odzwierciedleniem $U - DB$, ponieważ każda modyfikacja rekordu rozszerza zakres pokrywanych ciągów wprowadzając redundantne informacje. W [8] przedstawiony jest dowód że problem rekonstrukcji *DB* z *NDB* jest NP-trudny (każdą instancję 3-SAT można sprowadzić do *NDB*).

Powyższy algorytm w teorii jest zdolny do tworzenia trudnych instancji, ale nie posiada żadnych mechanizmów umożliwiających ingerencję w jego działanie – wynik jest zawsze losowy. Prawdopodobieństwo uzyskania trudnej do złamania kombinacji rekordów jest bardzo małe, co powoduje że w znacznej większości przypadków wynikowa *NDB* może być odwrócona niemal natychmiast przez współczesne solwery SAT, co pokazują w dalszych rozdziałach.

DB	U - DB	NDB
0000	0001	*001
0110	0011	00*1
0010	0100	0011
1101	0101	010*
	0111	0*11
	1000	1**0
	1001	10**
	1010	101*
	1011	**11
	1100	1*1*
	1110	1100
	1111	111*

Tabela 3.3. Rezultat działania algorytmu *Randomize_NDB*

3.3.3. Algorytm 0-*Hidden*

Następujące algorytmy powstały przez zastosowanie procedur generowania trudnych instancji SAT. Jedną z nich jest **0-*Hidden*** służąca do generacji formuł 3-SAT nie posiadających rozwiązania, co pozwala na testowanie solverów pod względem wykrywania braku spełnialności [10].

Algorytm 6: Algorytm 0-*Hidden*

Dane: l - liczba zmiennych, r - współczynnik ilości klauzul

Wynik: Zbiór klauzul 3CNF

```

1  $n = l * r$ 
2  $W = \{\}$ 
3 while  $|W| \neq n$  do
4     Wybierz 3 losowe zmienne
5     Stwórz klauzulę 3CNF używając wylosowanych zmiennych, z losowymi znakami
6     Dodaj klauzulę do zbioru wynikowego  $W$ 
7 end
```

Powyższy algorytm generuje formułę CNF, która z dużym prawdopodobieństwem nie jest spełnialna i jest trudna do rozwiązania przez solwery SAT [10, 11]. Modyfikując parametr r możemy wpłynąć na rozmiar formuły, wartość $r \approx 4.27$ jest wartością graniczną powyżej której problem prawie na pewno nie ma rozwiązania [10].

Z perspektywy Negatywnych Baz Danych można przyjąć że powstała formuła odpowiada zbiorowi $U - DB$, jeśli DB jest zbiorem pustym.

3.3.4. Algorytmy 1-*Hidden* i 2-*Hidden*

Algorytmy **1-*Hidden*** i **2-*Hidden*** są konstruowane podobnie jak **0-*Hidden***, z tą różnicą, że formuła wyjściowa ma odpowiednio co najmniej (i w znacznej większości dokładnie) jedno lub dwa rozwiązania.

W celu testowania możliwości solverów SAT co do znajdowania przypisania spełniającego daną formułę można generować losowe formuły (Algorytm 6) i odrzucać klauzule które przeczą ukrytemu rozwiązaniu A (**1-*Hidden***). Jednak spowoduje to, że rozkład losowy zostanie zaburzony i solwery mogą to „poczuć”, co doprowadzi je do ukrytego rozwiązania [11].

Aby temu przeciwdziałać można jednocześnie ukryć przypisania A oraz $\neg A$ (**2-*Hidden***), co spowoduje że algorytm przeszukujący będzie równoważnie „przyciągany” przez dwa przeciwne rozwiązania.

Powyższy schemat działania jest wykorzystany w procedurze **Q-*Hidden*** będącej rozszerzeniem tego konceptu.

3.3.5. Algorytm *Q-Hidden*

Metoda *Q-Hidden* do generacji trudnych formuł K-SAT zaproponowana w [10] rozszerza możliwości algorytmów z rozdziałów 3.3.3 i 3.3.4 przez dodanie parametru prawdopodobieństwa $q \in (0, 1)$. Każda wygenerowana klauzula składa się z k zmiennych których przypisanie pokrywa się z $t > 0$ literałów z prawdopodobieństwem q^t (gdzie t to numer kolejnego zgodnego literału). Oznacza to że im mniejsza wartość q tym bardziej formuła będzie wskazywać w przeciwnym kierunku niż ukryte rozwiązanie A .

Na podstawie tego schematu powstał algorytm mający na celu rozwiązanie problemów z algorytmami prefiksowym i Randomize_NDB tj. generowanie zbiorów rekordów które łatwo odwrócić za pomocą solverów SAT[12]

Algorytm 7: Algorytm *Q-Hidden*

Dane: s - ciąg bitowy, l - długość rekordu, r - wsp. ilości klauzul
 k - ilość ustalonych bitów w klauzuli, q - wsp. prawdopodobieństwa

Wynik: Zbiór rekordów NDB

```

1   $n = l * r$ 
2   $NDB = \{\}$ 
3  while  $|NDB| \neq n$  do
4       $\Upsilon = k$  różnych losowych pozycji w przedziale  $[1, l]$ 
5       $V =$  rekord  $NDB$  o długości  $l$  wypełniony znakami '*'
6      foreach  $i$  in  $\Upsilon$  do
7           $V[i] = s[i]$ 
8      end
9       $d = 0$ 
10     while  $d \neq 1$  do
11         foreach  $i$  in  $\Upsilon$  do
12              $p =$  losowa liczba rzeczywista z przedziału  $[0, 1]$ 
13             if  $q > p$  then
14                  $V[i] = \neg V[i]$ 
15                  $d = 1$ 
16             end
17         end
18     end
19     Dodaj rekord  $V$  do  $NDB$ 
20 end

```

W przeciwieństwie do poprzednich metod, za pomocą algorytmu *Q-Hidden* przy zastosowaniu odpowiednich parametrów q , r i k jest możliwe praktycznie stworzenie instancji NDB rozwiązywalnych

NDB		
*10*1	11**0	**100
11**0	**100	*000*
**100	*000*	0*01*
000	0*01*	*0*01
0*01*	*0*01	01*1*
*0*01	01*1*	1*0*1
01*1*	1*0*1	1**01

Tabela 3.4. Rezultat działania algorytmu *Q-Hidden* dla ciągu bitowego 10010 przy parametrach $k = 3$, $q = 0.5$ i $r = 4.2$

przez solwery SAT w czasie potęgowym tj. przez całkowite przeszukiwanie, co powoduje że dla odpowiednio dużych l odzyskanie zabezpieczonego ciągu jest niepraktyczne.

Takie podejście do problemu ma szereg konsekwencji. Pierwszą z nich jest to że wielkość otrzymanej bazy nie zależy od przechowywanego rekordu i można ją kontrolować za pomocą parametru r – generowane jest dokładnie $l * r$ rekordów. Nie ma jednak możliwości zawarcia w jednej instancji *NDB* więcej niż jednego ciągu bitowego - można zakodować jedynie zero lub jeden rekord.

W [12] razem z przedstawieniem algorytmu 7 opisany jest przykładowe zastosowanie tego schematu w systemach rzeczywistych wymagających znacznie większych ilości danych. Polega na przechowywaniu zbioru osobnych *NDB* dla każdego pozytywnego rekordu. Metoda zapytania do tak skonstruowanej bazy polega na sprawdzeniu po kolei wszystkich pojedynczych *NDB*. Jeżeli rekord nie pokrywa się z żadną formułą w jakiegokolwiek pod-bazie to przyjmuje się że znajduje się w bazie głównej.

Negatywne Bazy Danych nie przechowujące żadnych danych okazują się przydatne w takim rozwiązaniu ponieważ „puste” bazy stworzone przez algorytm *0-Hidden* (zmodyfikowany w celu generowania formuł o wymaganej liczbie literałów w klauzuli) są nierozróżnialne od tych wygenerowanych przez algorytm *Q-Hidden*. Pozwala to na ukrycie liczby rekordów pozytywnych.

Istotnym problemem jest fakt, że nie ma gwarancji, że rezultat działania tego algorytmu będzie pokrywał cały zbiór $U - \{s\}$, nawet powyżej wartości granicznej $r \approx 4.27$. Jedną z opcji jest dalsze zwiększenie parametru r żeby jeszcze bardziej zmniejszyć prawdopodobieństwo zawarcia nadmiarowych ciągów, ale wprowadza to dodatkowe, redundantne informacje co powoduje że czas potrzebny na odwrócenie *NDB* maleje. Dlatego wskazane jest ustawienie niskiej wartości r i dodanie do każdego rekordu dodatkowej informacji wskazującej na jego poprawność np. bitu parzystości, kodu CRC lub wyniku funkcji skrótu. W takiej sytuacji prawdopodobieństwo wystąpienia niechcianych danych maleje 2^c -krotnie, gdzie c oznacza liczbę dodatkowych bitów.

Rozkład wartości nadmiarowych ciągów bitowych nie jest jednostajny - dodatkowe wystąpienia będą bliskie s w odległości Hamminga (tj. poszczególne bity będą się różniły w niewielkiej ilości), dlatego

metoda CRC maksymalizująca odległość Hamminga dla podobnych napisów może być szczególnie efektywna. [12]

Porównanie poszczególnych metod przedstawiam w rozdziale 4.3.

3.3.6. Algorytm *K-Hidden*

K-Hidden jest modyfikacją algorytmu *Q-Hidden*. Zamiast pojedynczego parametru prawdopodobieństwa q stosuje się w nim wektor parametrów $\{p_1 \dots p_k\}$ w celu zwiększenia ilości stopni swobody algorytmu [13]. W ten sposób możemy dowolnie kontrolować dystrybucję różnych typów klauzul - w zależności od liczby pokrywających się bitów z ciągiem wejściowym.

Algorytm 8: Algorytm *K-Hidden*

Dane: s - ciąg bitowy, l - długość rekordu, r - wsp. ilości klauzul

k - ilość ustalonych bitów w klauzuli, $\{p_1 \dots p_k\}$ - wsp. prawdopodobieństwa

Wynik: Zbiór rekordów NDB

```

1  $n = l * r$ 
2  $NDB = \{\}$ 
3  $Q = \{Q_0, Q_1 \dots Q_k\} : Q_0 = 0, Q_i = p_1 + \dots + p_i$ 
4 while  $|NDB| \neq n$  do
5    $rnd = \text{losowa liczba rzeczywista z przedziału } (0, 1)$ 
6    $i = i : Q_{i-1} \leq rnd \leq Q_i$ 
7    $V = \text{rekord z } k \text{ ustalonymi bitami, pokrywający się z } s \text{ na } k - i \text{ losowych pozycjach}$ 
8   Dodaj rekord  $V$  do  $NDB$ 
9 end
```

Przed generacją każdego kolejnego rekordu losuje się liczbę z przedziału $(0, 1)$ i na podstawie jej i wektora parametrów p_i wybiera się odpowiedni typ rekordu - mający dokładnie i pokrywających się zmiennych gdzie i jest indeksem najmniejszego parametru większego od wylosowanej liczby.

Rezultat działania ma podobne cechy co wygenerowany przez *Q-Hidden* - możliwość zakodowania tylko jednego napisu i brak gwarancji pokrycia całego zbioru $U - \{s\}$.

W literaturze pojawia się także zaproponowany wcześniej algorytm *p-Hidden* będący szczególnym przypadkiem *K-Hidden* dla $k = 3$ [14]. Analogicznie dla przypadku gdy wektor parametrów p jest następujący:

$$p_i = \frac{\binom{k}{i} q^i}{(1+q)^k - 1}, i = 1 \dots k - 1$$

algorytm jest równoważny *Q-Hidden*. [13]

3.3.7. Algorytm hybrydowy

Algorytm hybrydowy jest połączeniem algorytmu prefikсового oraz *Q-Hidden* [15]. Ma na celu zapewnienie, że wygenerowana baza będzie zawierać całe uniwersum napisów bez jednego, ukrytego rekordu i jednocześnie będzie odporna na odwrócenie. Baza wynikowa jest równoważna formule 3-CNF.

Algorytm 9: Algorytm hybrydowy

Dane: s - ciąg bitowy, l - długość rekordu, r - wsp. ilości klauzul

q - wsp. prawdopodobieństwa

Wynik: Zbiór rekordów NDB

1 $NDB = \text{GenComplete}(s, l)$

2 $NDB = \text{MakeHardReverse}(NDB, s, l, r, q)$

Proces generacji składa się z dwóch etapów. Pierwszy, procedura *GenComplete*, tworzy kompletny zbiór $l + 4$ rekordów pokrywający całe $U - \{s\}$. Na początku tworzy $l - 2$ rekordy kolejno zaprzeczając każdemu bitowi oprócz dwóch ostatnich i losuje dwie dodatkowe pozycje przed nim ustawiając je zgodnie z s . Następnie generuje dwa rekordy dla różniącej się pozycji drugiej i cztery dla pierwszej.

NDB jest tworzona dla losowej permutacji napisu s a następnie każdy rekord jest przekształcany z powrotem przez permutację odwrotną w celu utrudnienia analizy przez pomieszenie kolejności bitów.

W kolejnym etapie tworzone są pozostałe $l * r - (l + 4)$ rekordy przez zastosowanie procedury *MakeHardReverse* (Algorytm 11) - równoznacznej *Q-Hidden* dla $k = 3$.

Wynik działania tej metody dla nieznanego napisu s jest nierozróżnialny od baz utworzonych przez algorytmy *0-Hidden*, *1-Hidden*, *2-Hidden*, *Q-Hidden* i *K-Hidden* przy $k = 3$ i identycznym r . Różnić się będą jedynie rozkładem pozycji ustalonych zgodnych i niezgodnych z ukrytym napisem.

11000
*1*01
1*01*
111**
100**
101**
0**00
0**10
0**01
0**11

Tabela 3.5. Rezultat działania procedury *GenComplete* dla $\pi(s) = 11000$ przed zastosowaniem permutacji odwrotnej na rekordach

Algorytm 10: GenComplete**Dane:** s - ciąg bitowy, l - długość rekordu**Wynik:** Zbiór rekordów NDB

```

1   $\pi$  = losowa permutacja o długości  $l$ 
2  for  $k = l$  to 3 do
3       $V$  = rekord  $NDB$  o długości  $l$  wypełniony symbolami '*'
4       $V[k] = \neg\pi(s)[k]$ 
5       $i, j$  = dwie różne liczby z przedziału  $[1, k]$ 
6       $V[i] = \pi(s)[i]$ 
7       $V[j] = \pi(s)[j]$ 
8      Dodaj rekord  $V$  do  $NDB$ 
9  end
10  $V$  = rekord  $NDB$  o długości  $l$  wypełniony symbolami '*'
11  $V[1] = \pi(s)[1]$ 
12  $V[2] = \neg\pi(s)[2]$ 
13  $i$  = losowa liczba z przedziału  $[3, l]$ 
14  $V[i] = 0$ , Dodaj rekord  $V$  do  $NDB$ 
15  $V[i] = 1$ , Dodaj rekord  $V$  do  $NDB$ 
16  $i$  = losowa liczba z przedziału  $[2, l]$ 
17 for  $b = 0$  to 1 do
18      $V$  = rekord  $NDB$  o długości  $l$  wypełniony symbolami '*'
19      $V[1] = \neg\pi(s)[1]$ 
20      $j$  = losowa liczba z przedziału  $[2, l]$ , różna od  $i$ 
21      $V[i] = b$ 
22      $V[j] = 0$ , Dodaj rekord  $V$  do  $NDB$ 
23      $V[j] = 1$ , Dodaj rekord  $V$  do  $NDB$ 
24 end
25 Zwróć  $\pi^{-1}(NDB)$ 

```

Algorytm 11: MakeHardReverse**Dane:** s - ciąg bitowy, l - długość rekordu, r - wsp. ilości klauzul q - wsp. prawdopodobieństwa**Wynik:** Zbiór rekordów NDB

```

1   $n = l * r$ 
2  while  $|NDB| \neq n$  do
3       $\Upsilon = 3$  różne losowe pozycje w przedziale  $[1, l]$ 
4       $V =$  rekord  $NDB$  o długości  $l$  wypełniony znakami '*'
5       $X =$  losowe przypisanie  $\in \{0, 1\}^3$ 
6       $u = 3$ 
7      foreach  $i = 1$  to 3 do
8           $V[\Upsilon[i]] = X[i]$ 
9          if  $V[\Upsilon[i]] \neq s[\Upsilon[i]]$  then
10              $u = u + 1$ 
11         end
12     end
13     if  $u \neq 0$  then
14          $p =$  losowa liczba rzeczywista z przedziału  $(0, 1)$ 
15         if  $q^u > p$  then
16             Dodaj rekord  $V$  do  $NDB$ 
17         end
18     end
19     Dodaj rekord  $V$  do  $NDB$ 
20 end
21 Zwróć powstałą  $NDB$ 

```

3.4. Wykorzystywanie solwerów SAT w celu uzyskania przeciwobrazu Negatywnej Bazy Danych

Problem znalezienia ukrytego ciągu w Negatywnej Bazie Danych jest równoważny SAT i zapis napisów nad alfabetem $\{0, 1, *\}$ można sprowadzić do CNF w czasie liniowym. Dla przykładu, poniższy zbiór rekordów można zinterpretować jako „Znajdź takie przypisanie wartościami $\{0, 1\}$, żeby dla każdego rekordu nie pokrywało się na co najmniej jednej pozycji”.

Zatem pomijając symbole $*$, zamieniając wartości $\{0, 1\}$ na odpowiadające literały, negując je i łącząc operatorem alternatywy oraz wstawiając operatory koniunkcji pomiędzy powstałe klauzule otrzymujemy postać CNF:

11*0
001*
*111
*101

Tabela 3.6. Przykładowy zbiór rekordów *NDB*

$$(\neg x_1 \vee \neg x_2 \vee x_4) \wedge$$

$$(x_1 \vee x_2 \vee \neg x_3) \wedge$$

$$(\neg x_2 \vee \neg x_3 \vee \neg x_4) \wedge$$

$$(\neg x_2 \vee x_3 \vee \neg x_4)$$

Metoda działania solwerów SAT nie polega na znalezieniu wszystkich rozwiązań, tylko na sprawdzeniu czy dana formuła jest spełnialna, dlatego w sytuacji próby odwrócenia baz wielorekordowych lub wygenerowanych z użyciem algorytmów niekompletnych atakujący potrzebuje dla każdego pojedynczego rozwiązania dodać dodatkową formułę przeczącą znalezionemu rekordowi i ponownie uruchomić solwer.

W związku z faktem, że uzyskanie dostępu do nawet pojedynczego rekordu w systemach uwierzytelniania jest niedopuszczalne, testy są przeprowadzone pod kątem znalezienia pierwszego rozwiązania.

4. Testy algorytmów

4.1. Opis testowania za pomocą solverów SAT

Do testów *NDB* użyte zostały zChaff oraz WalkSAT – najbardziej popularny solver kompletny i niekompletny. Wszystkie testy zostały przeprowadzone za pomocą aplikacji konsolowej napisanej w języku C++17 z wykorzystaniem biblioteki *boost* do wygenerowania opcji programu i ułatwienia operacji na napisach.

```
$ ./NDBGenerator --help
Using seed: 1607338934763313530
Allowed commands:
-r [ --record-count-ratio ] arg Record count parameter
-q [ --probability-ratio ] arg Probability parameter
-p [ --probability-ratios ] arg Probability parameters
-l [ --record-length ] arg Record length
-c [ --record-count ] arg Record count
-k [ --specified-bits ] arg Specified bits count
-o [ --output-file ] arg Output file
-f [ --format ] arg Output format (dimacs | ndb)
-s [ --superfluous ] Don't generate file, check for superfluous
strings
--batch Enable batch mode
--repeat arg Specify times to repeat test
--checksum-bits arg Checksum bits count
--db-record arg Specify DB record
--distribution Run distribution test
-h [ --help ] Produce help message
--solve-tests Run solve tests
--gen-time-tests Run generation time tests
--cutoff arg Specify WalkSAT cutoff
--time-limit arg Specify zChaff time limit
--solver arg Specify solver (zchaff | walksat)
-a [ --algorithm ] arg Set algorithm (Prefix | Randomized |
RandomizedOld | 0-Hidden | Q-Hidden |
K-Hidden | Hybrid)
```

Rys. 4.1. Opcje aplikacji testującej

Przy wywołaniu programu użytkownik może ustawić szereg wartości dla każdego dostępnego parametru i wybrać pożądany algorytm oraz typ testu. Aplikacja następnie generuje losową *NDB* dla każdej kombinacji parametrów, przeprowadza test oraz zapisuje wynik do pliku w formacie csv.

Możliwe jest także wygenerowanie pliku Negatywnej Bazy Danych w formacie *ndb* tj. plik tekstowy zawierający napisy nad alfabetem $\{0, 1, *\}$, lub *dimacs* - domyślnym formacie wykorzystywanym przez solwery SAT.

```
c Przykładowy plik cnf
p cnf 3 2
-1 -3 0
-2 3 1 0
```

Listing 4.1. Przykładowy plik formatu *dimacs*

Zawartość składa się z komentarzy (linie zaczynające się znakiem „c”), definicji problemu (linia zaczynająca się znakiem „p”) i klauzul logicznych.

W tym przypadku opisywany jest problem CNF o trzech zmiennych i dwóch klauzulach reprezentujący formułę $(\neg x_1 \vee \neg x_3) \wedge (\neg x_2 \vee x_3 \vee x_1)$.

Aplikacja jest zintegrowana z napisaną w języku C implementacją algorytmu WalkSAT przedstawioną przez Uniwersytet w Rochester [16] oraz implementacją zChaff w języku C++ udostępnioną przez Uniwersytet w Princeton [17].

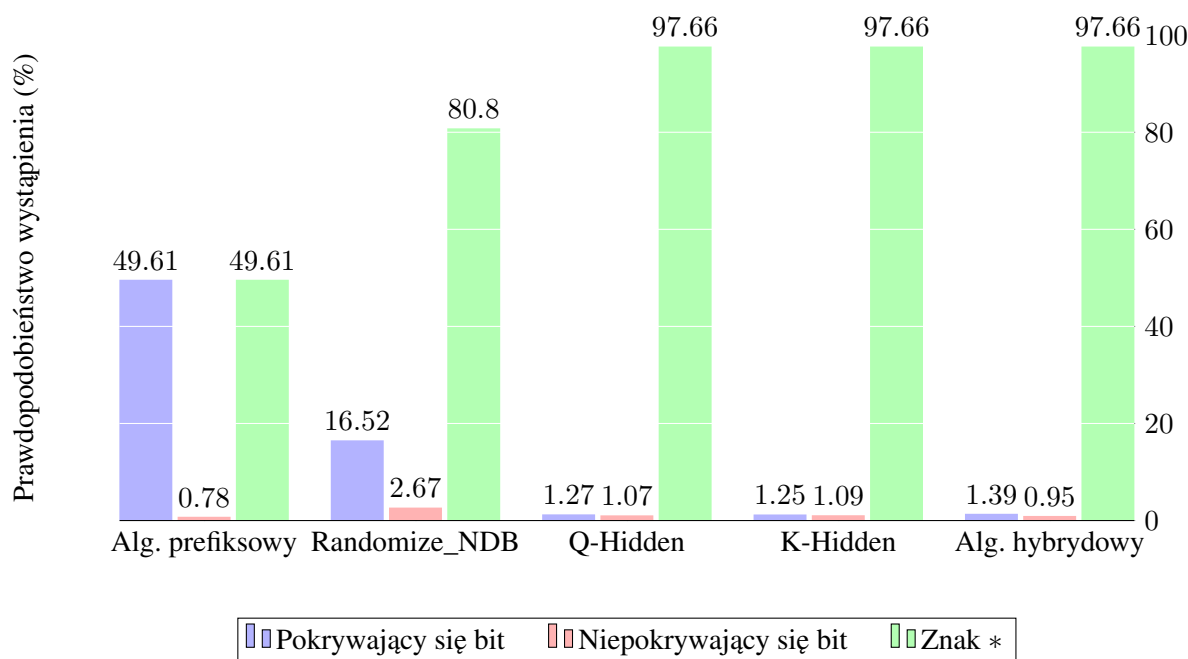
Oba programy posiadają interfejs wiersza poleceń i przyjmują pliki w formacie *dimacs*. Z powodu potrzeby testowania znacznej ilości automatycznie generowanych formuł aplikacja testowa wywołuje procedury rozwiązujące bezpośrednio z kodu. W przypadku solwera zChaff wymagane było jedynie załączenie pliku nagłówkowego i zalinkowaniu zbudowanej biblioteki, jednak przy integrowaniu solwera WalkSAT konieczna była minimalna modyfikacja plików źródłowych w celu umożliwienia wywołania głównej funkcji oraz zresetowania stanu po próbie rozwiązania każdej formuły.

4.2. Rozkład statystyczny *NDB*

Negatywne Bazy Danych wygenerowane przez poszczególne algorytmy mogą różnić się znacząco. Jednym ze sposobów ich porównania jest przedstawienie prawdopodobieństw pojawienia się określonych typów literałów w rekordach *NDB* tj. bitów pokrywających się z ukrytym napisem, bitów zaneigowanych oraz znaków specjalnych „*”. Wynik działania testu rozkładu znajduje się na wykresie 4.2.

Algorytm prefiksowy generuje rekordy, w których znak może być odwrócony tylko na jednej pozycji – zaraz przed serią znaków „*”. Pozostałe możliwości są podzielone równo po połowie. W *Randomize_NDB* znaczna część bitów jest zamieniana na „*”, dodatkowo ustawiana jest niewielka liczba znaków na odwrócone.

Pozostałe algorytmy generują rekordy składające się z bardzo niewielu pozycji ustalonych, a ich wzajemne proporcje są modyfikowalne przez parametry q lub p . Analizę ich wpływu na złożoność wygenerowanej bazy przedstawiam w rozdziałach 4.4.3, 4.4.4 i 4.4.5.



Rys. 4.2. Rozkład prawdopodobieństwa pojedynczych pozycji w wygenerowanych rekordach *NDB*. Dane zgromadzone na podstawie 1000 generacji dla każdego algorytmu. Użyte parametry: $l = 128$, $k = 3$, $q = 0.3$, $p = \{0.7, 0.2, 0.1\}$, $c = 1$, $r = 4.5$

4.3. Testy metod weryfikujących poprawność rekordów pozytywnych

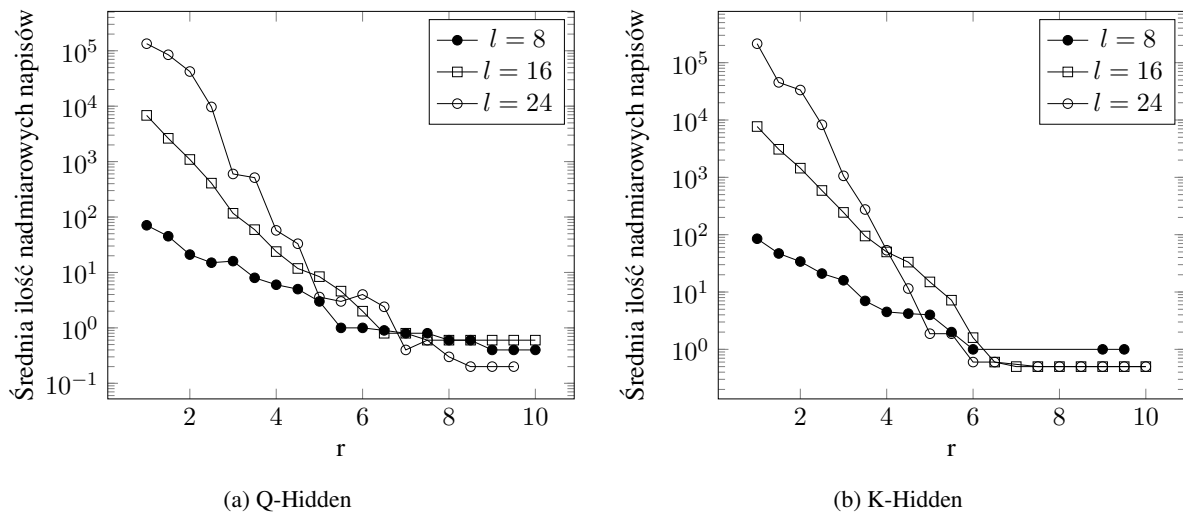
Algorytmy generacji *NDB* *Q-Hidden* oraz *K-Hidden* nie gwarantują pokrycia całego zbioru $U - DB$. Konieczne jest użycie odpowiedniej sumy kontrolnej. Po znalezieniu rozwiązania porównywany jest sufix rekordu o określonej długości z wynikiem wykorzystanej funkcji na pozostałej części ciągu bitowego. Jeżeli wynik się zgadza napis jest uznawany za należący do *DB*.

Zostały przetestowane następujące funkcje sumy kontrolnej:

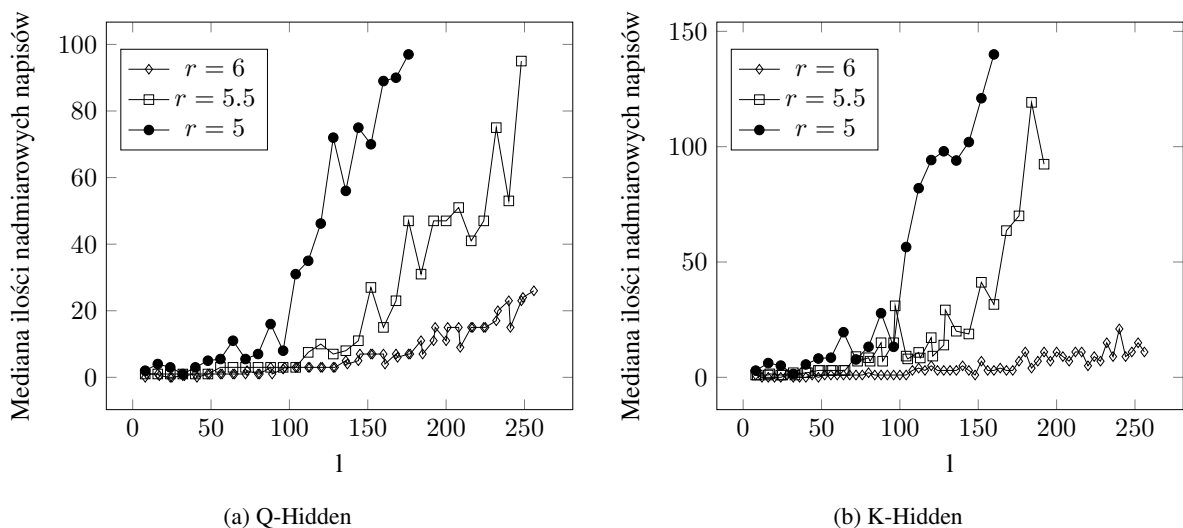
- Bit parzystości (1 bit)
- CRC-8 (8 bitów)
- CRC-16 (16 bitów)
- CRC-32 (32 bity)
- MD5 (128 bitów)

4.3.1. Wpływ parametrów generacji na ilość niepożądanych ukrytych ciągów bitowych

Jak wynika z testów, których wyniki są umieszczone na wykresach 4.3a i 4.3b ilość nadmiarowych napisów ukrytych maleje wykładniczo wraz z wzrostem parametru r . Dla ilości rekordów pięciokrotnie większej niż długość napisu zachowują się jedynie pojedyncze niepożądane ciągi.



Rys. 4.3. Zależność ilości nadmiarowych napisów od współczynnika ilości rekordów r dla *NDB* wygenerowanych przez algorytm *Q-Hidden* i *K-Hidden* o parametrach $q = 0.4 / p = \{0.7, 0.2, 0.1\}$ i $k = 3$.



Rys. 4.4. Zależność ilości nadmiarowych napisów od ilości rekordów l dla *NDB* wygenerowanych przez algorytm *Q-Hidden* i *K-Hidden* o parametrach $q = 0.3 / p = \{0.7, 0.2, 0.1\}$ i $k = 3$.

Długość rekordu także ma znaczny wpływ na ich ilość, zwłaszcza dla małych wartości r , jednak przeprowadzanie testów dla całego przedziału $r \in [1, 10]$ dla $l > 24$ jest niepraktyczne ze względu na złożoność obliczeniową.

Wraz ze wzrostem parametru r zwiększa się wartość progowa długości rekordu, której przekroczenie powoduje znaczne zwiększenie tempa przyrostu ilości napisów nienależących do DB . Jak wynika z wykresów 4.4a i 4.5 dla wartości $r = 5$ pochodna funkcji wzrasta przy $l \approx 100$ a dla $r = 5.5$ przy $l \approx 150$.

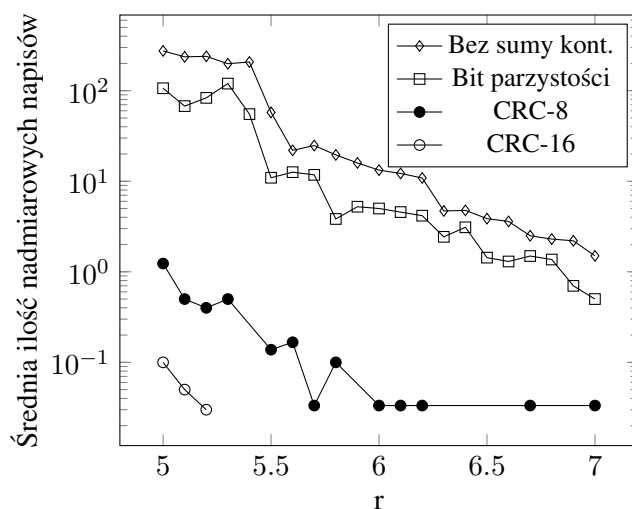
Z powodu znacznej fluktuacji wyników aby przedstawić dokładną tendencję konieczne było zastosowanie mediany zamiast wartości średniej.

Negatywne Bazy Danych wygenerowane przez algorytm *Q-Hidden* i *K-Hidden* nie różnią się znacząco od siebie w kontekście testu.

4.3.2. Skuteczność funkcji sum kontrolnych

Wynik porównania różnych metod sumy kontrolnej pokrywa się z intuicją, według której każdy dodatkowy bit zmniejsza ilość ciągów bitowych nienależących do DB o połowę. Wynika z tego, że funkcja skrótu o długości 32 bitów, jest wystarczająca, żeby zapewnić spójność bazy danych przy zastosowaniach w których nie trzeba przechowywać danych krytycznych.

W systemach uwierzytelniania zalecane jest jednak użycie większej wartości w celu zredukowania szansy konfliktu danych do praktycznie niemożliwej.

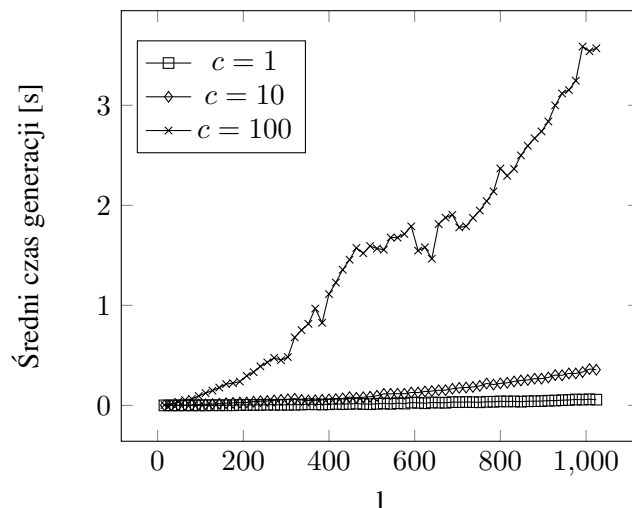


Rys. 4.5. Zależność ilości nadmiarowych napisów od współczynnika ilości rekordów r i metody sumy kontrolnej. NDB zostały wygenerowane przez algorytm *K-Hidden* o parametrach $p = \{0.7, 0.2, 0.1\}$ i $k = 3$ oraz *Q-Hidden* o parametrach $q = 0.3$ i $k = 3$. Dla CRC-32 oraz MD5 nie zostały wykryte żadne nadmiarowe napisy.

4.4. Testy algorytmów generacji *NDB*

4.4.1. Algorytm prefiksowy

Algorytm prefiksowy jest bardzo prostą metodą generacji *NDB*, dlatego odwrócenie rezultatu jej działania nie stanowi problemu. Sposób na uzyskanie zbioru *DB* bez użycia solwera SAT został zaprezentowany w algorytmie 2. Generacja ma stosunkową niską złożoność czasową, nawet dla bardzo długich ciągów, jednak zawarcie wielu rekordów *DB* może ją znacznie spowolnić¹.



Rys. 4.6. Zależność czasu generacji *NDB* za pomocą algorytmu prefiksowego od długości kodowanego ciągu.

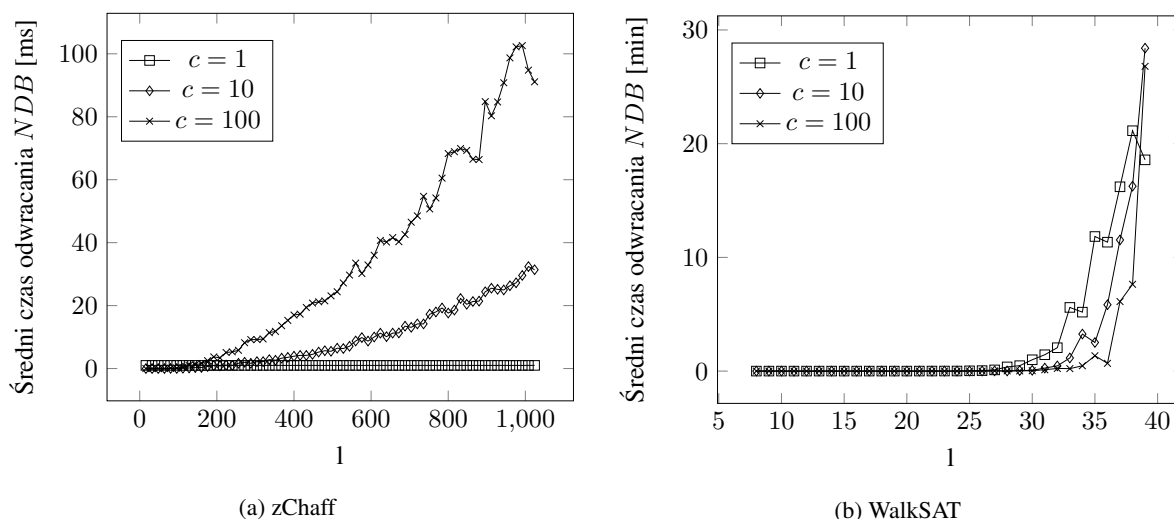
Kompletny solwer zChaff odwraca *NDB* z jednym rekordem pozytywnym niemal natychmiast – dla każdego przypadku została podjęta tylko jedna decyzja. Przy zawarciu większej ilości napisów znalezienie jednego z nich staje się trudniejsze, jednak jest to nadal możliwe w czasie liniowym.

Natomiast dla solwera WalkSAT otrzymana struktura CNF jest problematyczna – procedura degradowuje się w kierunku losowego przeszukiwania o wykładniczej złożoności obliczeniowej. Już dla stu napisów o długości 40 bitów metoda kończy działanie z powodu przekroczenia ustawionej na potrzeby testów wartości granicznej 10 prób po 10^9 odwróceń przypisań.

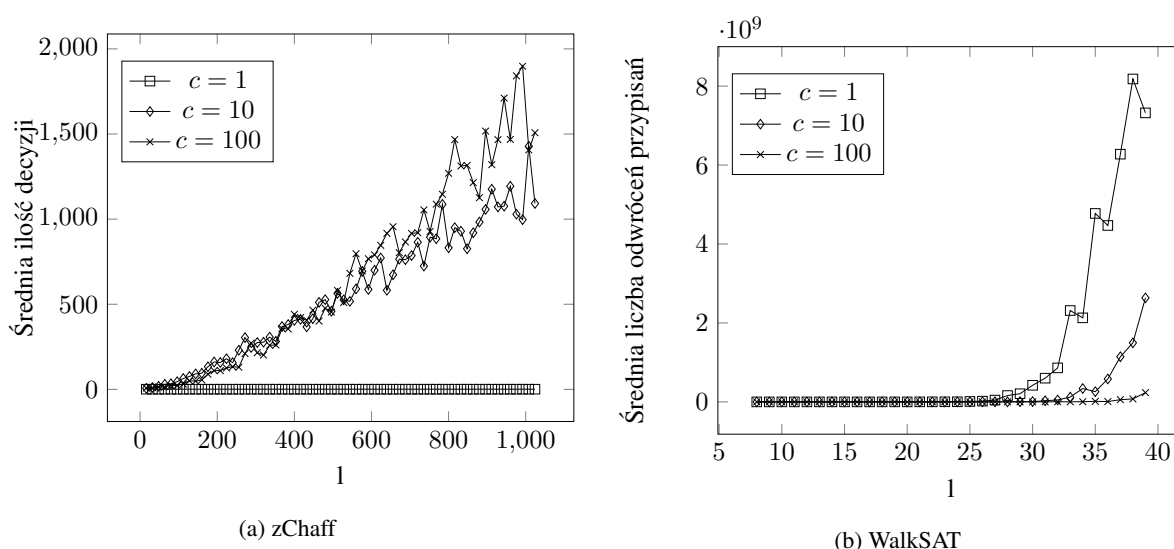
4.4.2. Algorytm *Randomize_NDB*

Algorytm *Randomize_NDB* generuje bazy danych o znacznie większym rozmiarze niż algorytm prefiksowy. Już dla 25 rekordów, każdy o długości co najmniej 300 bitów, czas trwania procedury generacyjnej przekracza godzinę, podczas, gdy czas rozwiązywania bazy o takich parametrach zajmuje solwerowi zChaff niecałe dwie sekundy. Dla ukrywania pojedynczych rekordów procedura działa niewiele wolniej niż algorytm prefiksowy, ale wynikowa *NDB* będzie bardzo prosta do złamania.

¹Testy czasowe algorytmów zostały przeprowadzone przez wyliczenie czasu procesora podczas działania określonej procedury na komputerze wyposażonym w procesor Intel i5-7200U 3.1 GHz i 16 GB pamięci RAM.



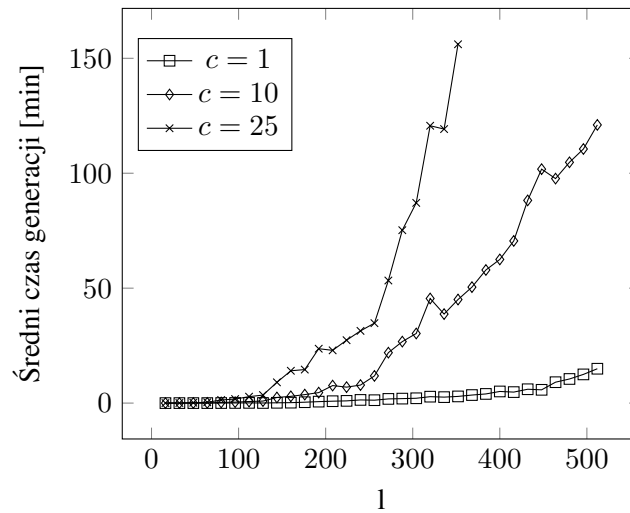
Rys. 4.7. Zależność czasu koniecznego do znalezienia pierwszego rozwiązania od długości ciągu i ilości rekordów dla *NDB* wygenerowanych za pomocą algorytmu prefiksowego.



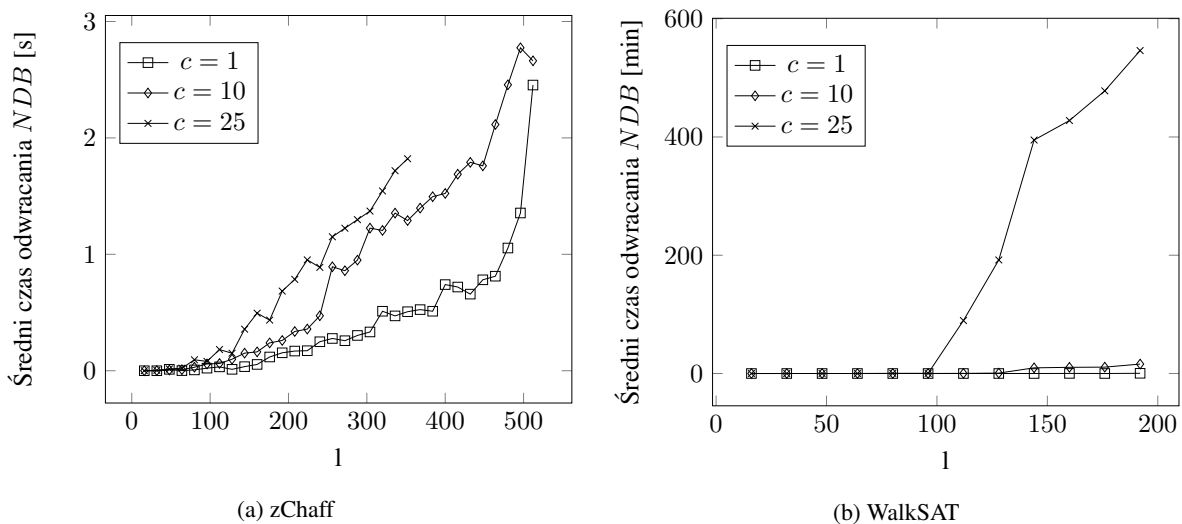
Rys. 4.8. Zależność liczby decyzji/odwróceń przypisań od długości ciągu i ilości rekordów dla *NDB* wygenerowanych za pomocą algorytmu prefiksowego.

Złożoność otrzymanej bazy zależy głównie od ilości zawartych rekordów pozytywnych, ale czas generacji rośnie dużo szybciej niż trudność problemu SAT. z wykresu 4.10a na pierwszy rzut oka wynika, że dla większych wartości l czas rozwiązania znacznie wzrasta, ale jest to spowodowane procesem przetwarzania wstępnego solwera zChaff – dla wszystkich pokazanych przykładów liczba podjętych decyzji nie przekracza pięciu.

Solwer WalkSAT podobnie jak w przypadku algorytmu prefiksowego wypada znacznie gorzej niż zChaff, jednak tym razem potrafi w krótkim czasie odwrócić *NDB* przy $l = 100$ i niewielkiej ilości rekordów pozytywnych, jednak wraz ze wzrostem c złożoność obliczeniowa zwiększa się drastycznie.



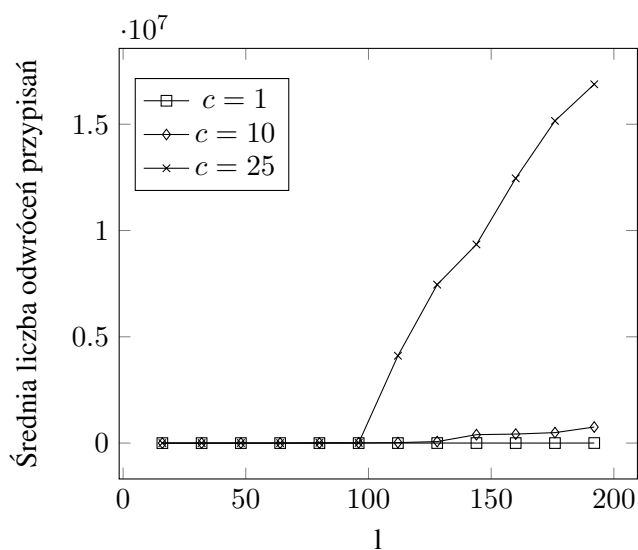
Rys. 4.9. Zależność czasu generacji *NDB* za pomocą algorytmu *Randomize_NDB* od długości kodowanego ciągu.



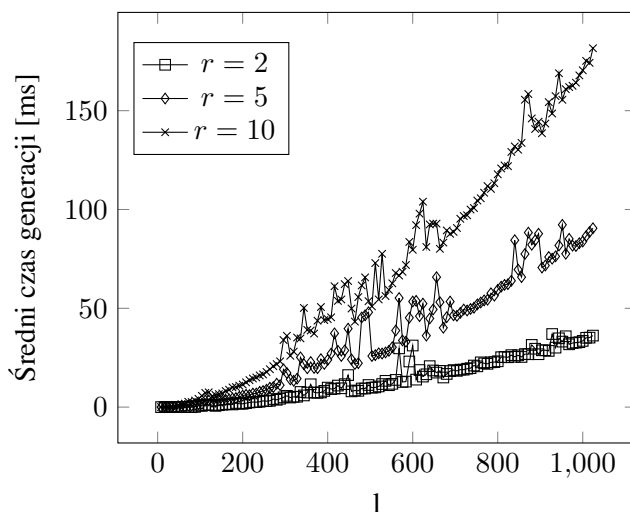
Rys. 4.10. Zależność czasu koniecznego do znalezienia pierwszego rozwiązania od długości ciągu i ilości rekordów dla *NDB* wygenerowanych za pomocą algorytmu *Randomize_NDB*.

4.4.3. Algorytm *Q-Hidden*

W przeciwieństwie do poprzednich algorytmów, *Q-Hidden* generuje *NDB*, które mogą przechowywać jedynie jeden rekord pozytywny. Nie jest to jednak problemem dla systemów przechowujących znaczne ilości danych, gdyż pomimo konieczności tworzenia osobnej bazy dla każdego rekordu, takie podejście zapewnia dużo krótsze czasy generacji i dużo mniej potrzebnej przestrzeni dyskowej niż algorytmy prefiksowy i *Randomize_NDB*. Znacznie zredukowanie liczby rekordów negatywnych powoduje też skrócenie czasu zapytań.



Rys. 4.11. Zależność liczby odwróceń przypisań wykonanych przez solver WalkSAT od długości ciągu i ilości rekordów dla *NDB* wygenerowanych za pomocą algorytmu *Randomize_NDB*.

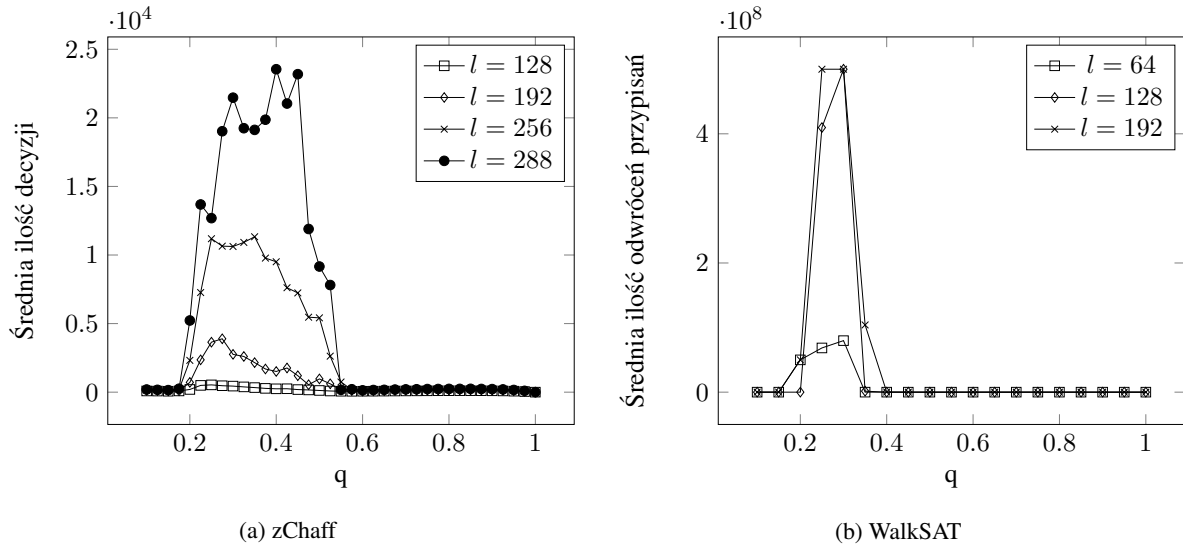


Rys. 4.12. Zależność czasu generacji *NDB* za pomocą algorytmu *Q-Hidden* od długości kodowanego ciągu.

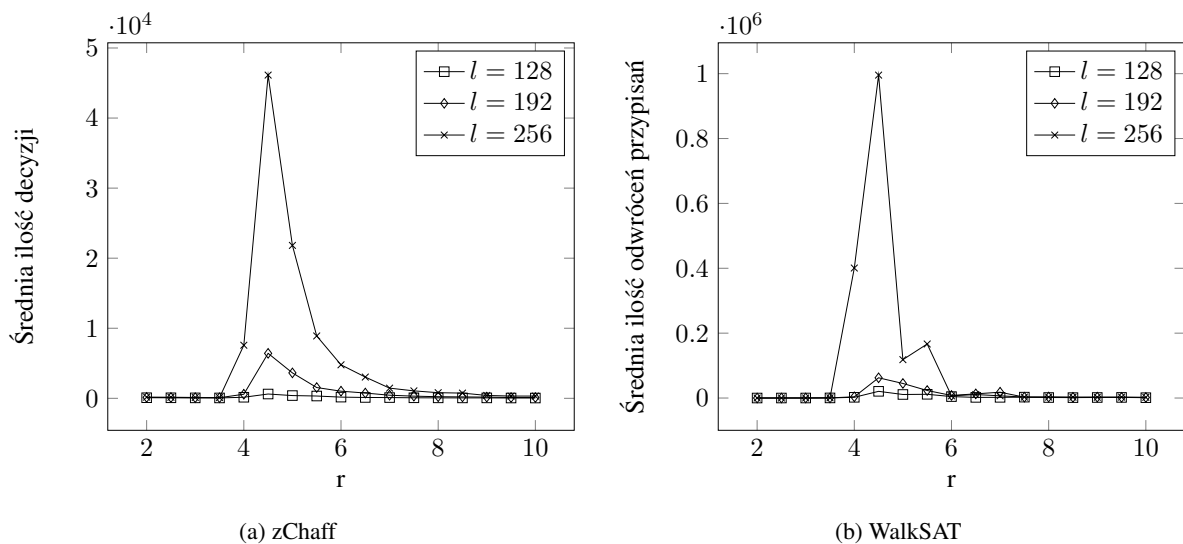
Algorytm *Q-Hidden* tworzy bazy o dowolnie konfigurowanej, stałej liczbie rekordów. Generacja *NDB* jest bardzo szybka i dokonuje się w czasie liniowym od iloczynu parametrów r oraz l .

Głównymi czynnikami regulującymi złożoność *NDB* wygenerowanej przez *Q-Hidden* są parametry q – kontrolujący jak często ustalone literały w rekordach negatywnych będą się pokrywać z ukrytym napisem, oraz r – regulujący wielkość otrzymanej bazy. Wyniki testu badającego wpływ tego parametru są przedstawione na wykresach 4.13a i 4.13b. Wynika z nich, że przedział zapewniający największe bezpieczeństwo zawiera się między $q = 0.35$ a $q = 0.5$

Parametr wielkości generowanej bazy r musi być dostatecznie duży, żeby pokryta została jak największa część $U - DB$, oraz jednocześnie na tyle niewielki, żeby nie zawrzeć nadmiarowych informacji, które mogą zostać wykorzystane przez solver SAT. Eksperyment, którego wynik jest przedstawiony na wykresach 4.14a oraz 4.14b dowodzi, że najbardziej optymalną wartością jest $r = 4.5$.



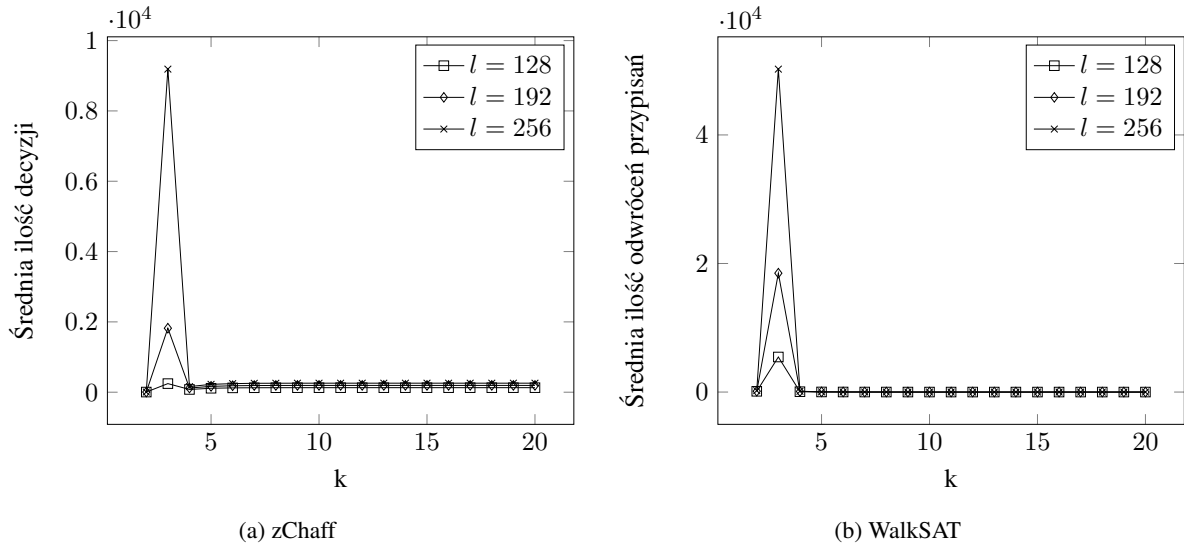
Rys. 4.13. Zależność liczby decyzji/odwróceń przypisań potrzebnych do znalezienia pierwszego rozwiązania od parametru q dla *NDB* wygenerowanych za pomocą algorytmu *Q-Hidden* przy parametrach $r = 5.5$ i $k = 3$.



Rys. 4.14. Zależność liczby decyzji/odwróceń przypisań potrzebnych do znalezienia pierwszego rozwiązania od parametru r dla *NDB* wygenerowanych za pomocą algorytmu *Q-Hidden* przy parametrach $k = 3$ i $q = 0.4$.

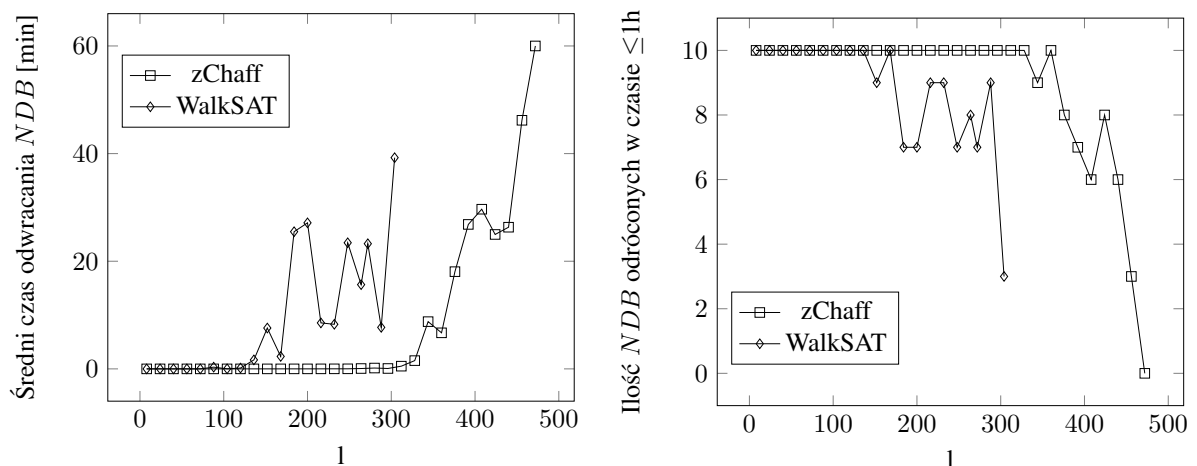
Algorytm *Q-Hidden* bazuje na metodach *1-Hidden* oraz *2-Hidden*, których wynikiem jest formuła 3-CNF, jednak pozwala na wybranie dowolnej stałej k w celu stworzenia problemu k -CNF. Analiza

eksperymentalna wykazała, że domyślna wartość $k = 3$ jest jednocześnie jedyną możliwą wartością z punktu widzenia bezpieczeństwa. Wykresy 4.15a i 4.15b pokazują, że każda inna wartość sprawia, że *NDB* jest odwracana niemal natychmiast.



Rys. 4.15. Zależność liczby decyzji/odwróceń przypisań potrzebnych do znalezienia pierwszego rozwiązania od parametru k dla *NDB* wygenerowanych za pomocą algorytmu *Q-Hidden* przy parametrach $r = 5.5$ i $q = 0.4$. Szczyt zachodzi dla parametru $k = 3$

Po wykonaniu powyższej analizy można stwierdzić, że optymalne parametry dla tego algorytmu to: $r = 4.5$, $k = 3$ oraz $q = 0.3$.



(a) Zależność czasu potrzebnego na odwrócenie *NDB* od długości rekordu.

(b) Zależność liczby odwróconych *NDB* w czasie nie większym niż jedna godzina od długości rekordu.

Rys. 4.16. Zależność trudności do odwrócenia od długości ukrytego rekordu. Testy przeprowadzone na podstawie dziesięciu *NDB* dla każdej wartości l , wygenerowanych przez algorytm *Q-Hidden* o parametrach: $r = 4.5$, $q = 0.3$ i $k = 3$.

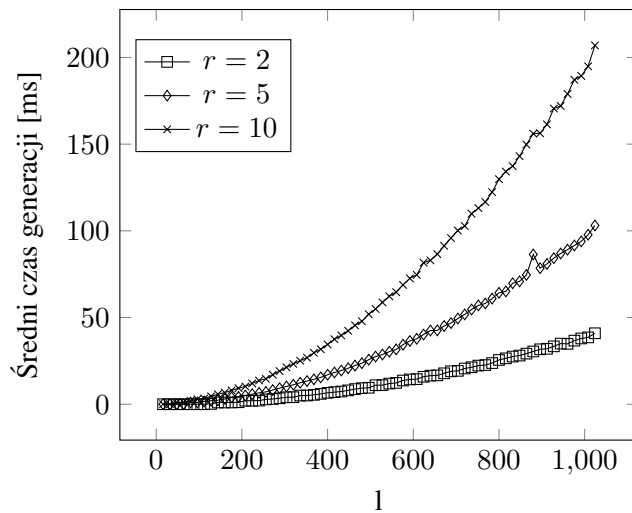
Po przeprowadzeniu testów analizujących wpływ długości rekordu na złożoność otrzymanej bazy (wykresy 4.16a i 4.16b) można zauważyć pojawienie się złożoności wykładniczej. Zmiany są bardzo stabilne dla solwera zChaff, nagły skok w czasie rozwiązywania następuje przy wartości ok. 350 bitów. Przy WalkSAT trudności pojawiają się dużo wcześniej, ale zmiana jest nieregularna. Dla pojedynczej wartości l czasy odwracania mogą wynosić kilka sekund lub ponad godzinę tj. do osiągnięcia limitu operacji (10 prób po $2 \cdot 10^9$ odwróceń), ale łatwe instancje pojawiają się coraz rzadziej wraz ze wzrostem długości rekordu.

Z powyższej analizy można wywnioskować, że tak wygenerowane *NDB* dla rekordów o długości od 600-700 bitów mogą oferować satysfakcjonującą odporność na odwrócenie.

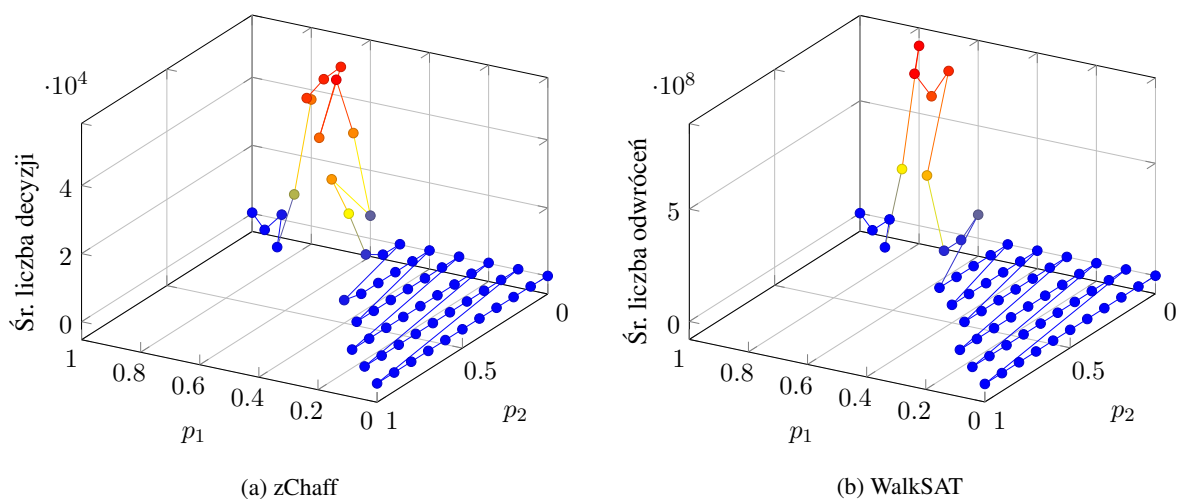
4.4.4. Algorytm *K-Hidden*

Jedyną różnicą między algorytmem *K-Hidden* a *Q-Hidden* jest możliwość dokładnego zdefiniowania z jakim prawdopodobieństwem powinny generować się rekordy *NDB* z określoną liczbą pokrywających się bitów.

Czas generacji jest prawie identyczny jak w *Q-Hidden*, z tą różnicą, że wzrost wraz ze zwiększającą się wartością $l * r$ jest dużo bardziej stabilny ponieważ podczas generacji każdego rekordu wiadome jest ile bitów należy zanegować – nie ma potrzeby odrzucania sytuacji w której wszystkie k literałów pokrywałyby się z kodowanym napisem.



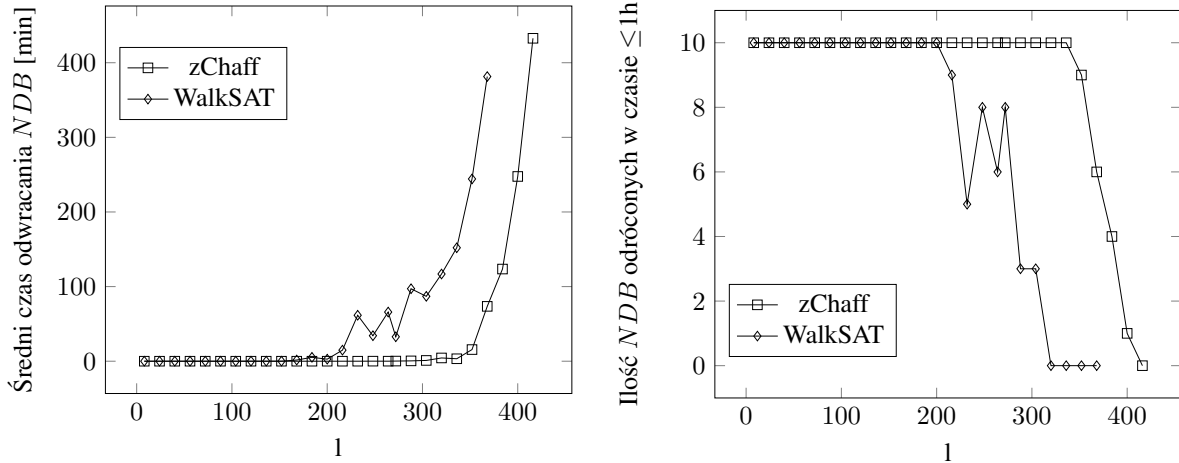
Rys. 4.17. Zależność czasu generacji bazy za pomocą algorytmu *K-Hidden* od długości kodowanego ciągu.



Rys. 4.18. Zależność liczby operacji potrzebnych do znalezienia pierwszego rozwiązania od wektora parametrów p dla *NDB* wygenerowanej za pomocą algorytmu *K-Hidden* przy parametrach $k = 3$, $r = 4.5$ oraz $l = 256$.

Po przeszukaniu dziedziny wektora p dla $k = 3$ pod kątem złożoności powstałej *NDB* (wykres 4.18a i 4.18b) jako najbardziej optymalny wyłonił się punkt $p = \{0.7, 0, 0.3\}$ – przy którym oba testowane solwery potrzebowały średnio najwięcej operacji.

Test został również przeprowadzony dla $k = 4$, ale rezultat okazał się analogiczny do jego odpowiednika w *Q-Hidden* – każda wartość $k \neq 3$ powoduje strywializowanie problemu.



(a) Zależność czasu potrzebnego na odwrócenie *NDB* od długości rekordu.

(b) Zależność liczby odwróconych *NDB* w czasie nie większym niż jedna godzina od długości rekordu.

Rys. 4.19. Zależność trudności do odwrócenia od długości ukrytego rekordu. Testy przeprowadzone na podstawie dziesięciu *NDB* dla każdej wartości l , wygenerowanych przez algorytm *K-Hidden* o parametrach: $r = 4.5$, $p = \{0.7, 0, 0.3\}$ i $k = 3$.

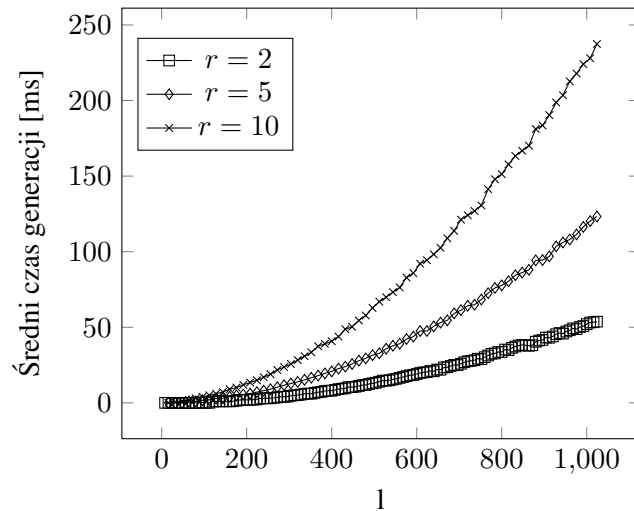
Łącząc wnioski z analizy algorytmu *Q-Hidden* i testów wpływu wektora p można stwierdzić, że zestaw parametrów $r = 4.5$, $p = \{0.7, 0, 0.3\}$ i $k = 3$ jest optymalny dla tej metody. Ostateczny test, przedstawiony na wykresach 4.19a i 4.19b wykazuje istnienie wykładniczej złożoności obliczeniowej wraz ze wzrostem l . Oznacza to, że tak skonstruowane *NDB* o długości rekordu większej niż 500 bitów oferują znaczną ochronę przed odwróceniem.

4.4.5. Algorytm hybrydowy

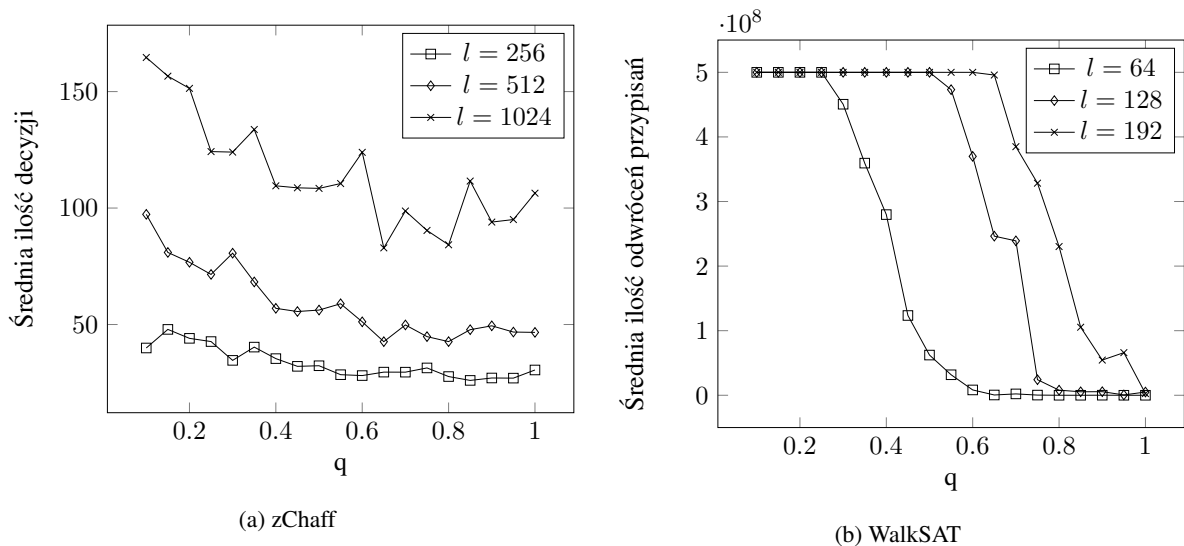
Algorytm hybrydowy jest połączeniem *Q-Hidden* i zmodyfikowanego algorytmu prefiksowego. Powstała baza jest konkatencją kompletnej *NDB* i wyniku algorytmu *Q-Hidden* z zmniejszoną ilością rekordów w celu poprawnego działania parametru r .

Niekompletność jest znaczną wadą przetestowanych wcześniej optymalnych metod generacji, dlatego istnienie bezpiecznego algorytmu gwarantującego pokrycie całego zbioru $U - \{s\}$ okazałoby się przydatne w systemach uwierzytelniania.

Z testów wynika jednak, że pomimo znacznej odporności na odwrócenie przez solver WalkSAT, to podejście posiada te same wady co algorytm prefiksowy. Podczas analizy eksperymentalnej za pomocą solwera zChaff nie znaleziono żadnej kombinacji parametrów pozwalających na uzyskanie wykładniczej



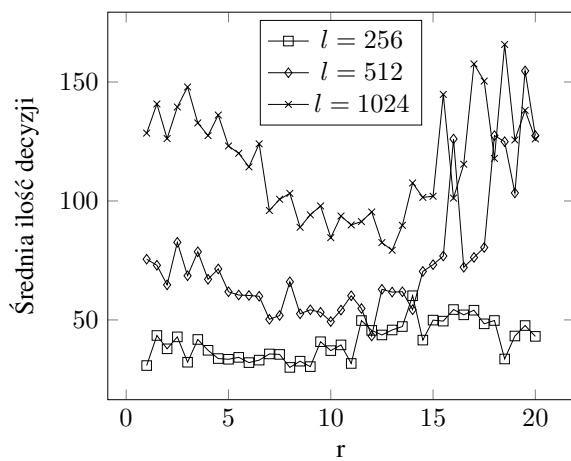
Rys. 4.20. Zależność czasu generacji bazy za pomocą algorytmu hybrydowego od długości kodowanego ciągu.



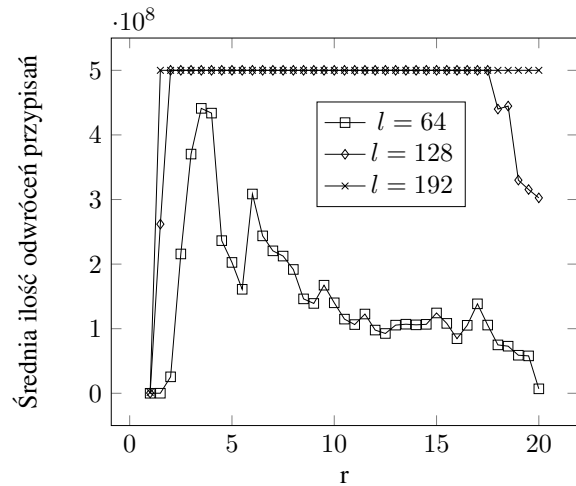
Rys. 4.21. Zależność liczby decyzji / odwróceń przypisań potrzebnych do znalezienia pierwszego rozwiązania od parametru q dla *NDB* wygenerowanych za pomocą algorytmu hybrydowego przy parametrze $r = 4.5$.

złożoności obliczeniowej. Jest to najprawdopodobniej spowodowane faktem, że w zbiorze rekordów *NDB* jest ukryty łatwy do odwrócenia podzbiór. Jedną z poszlak wskazujących na to jest niewielki wpływ parametrów q i r na ilość wykonanych decyzji przed znalezieniem rozwiązania. Żadne tendencje odnalezione podczas testów *Q-Hidden* i *K-Hidden* nie występują w tym przypadku, co potwierdza że z punktu widzenia solwera *zChaff* „bezpieczna” część *NDB* ma niewielki wpływ na zadany problem SAT.

Końcowy test tego algorytmu pokazany na wykresach 4.23a i 4.23b udowadnia, że tak wygenerowany problem SAT pozostaje łatwy nawet dla bardzo dużych długości rekordów. Każda przetestowana



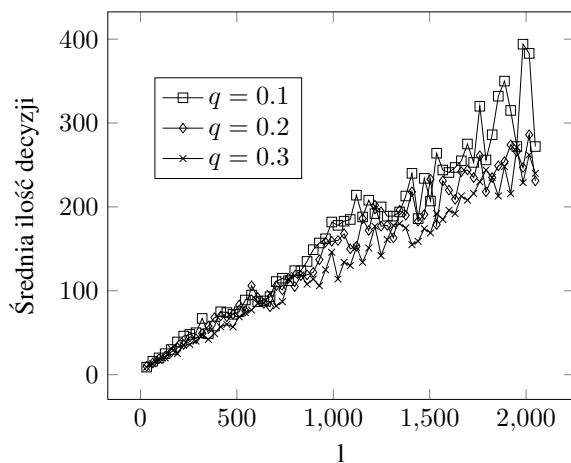
(a) zChaff



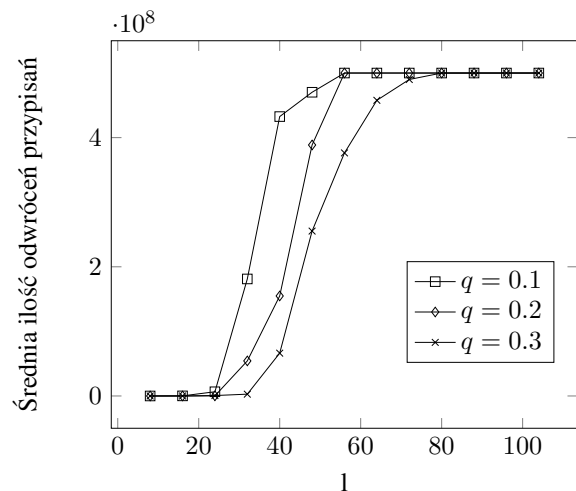
(b) WalkSAT

Rys. 4.22. Zależność liczby decyzji / odwróceń przypisań potrzebnych do znalezienia pierwszego rozwiązania od parametru r dla *NDB* wygenerowanej za pomocą algorytmu hybrydowego przy parametrze $q = 0.3$.

NDB została odwrócona w mniej niż dwie milisekundy przez solver zChaff. Przy WalkSAT już dla bardzo małych długości rekordów został osiągnięty limit $5 \cdot 10^7$ odwróceń występujący po ok. 30 minutach.



(a) zChaff



(b) WalkSAT

Rys. 4.23. Zależność liczby decyzji/odwróceń przypisań potrzebnych do znalezienia pierwszego rozwiązania od parametru l dla *NDB* wygenerowanej za pomocą algorytmu hybrydowego przy parametrze $r = 4.5$.

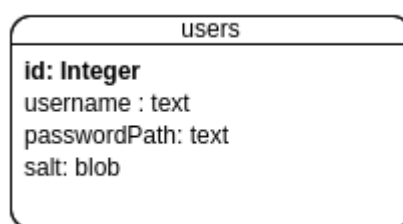
5. Implementacja systemu uwierzytelniania

W tym rozdziale przedstawiam system uwierzytelniania wykorzystujący *NDB* jako formę składowania skrótu hasła. Aplikacja nie składa się żadnych dodatkowych danych użytkownika, lecz może posłużyć jako baza dla systemów o dokładniej sprecyzowanym zastosowaniu.

Implementacja jest napisana w języku C++17 z użyciem biblioteki Qt, pozwalającej na łatwe projektowanie interfejsu użytkownika oraz dostarczającej wszystkie wymagane procedury kryptograficzne.

5.1. Reprezentacja danych

Część danych użytkownika pozwalających na zalogowanie się do systemu znajdują się w zwykłej, relacyjnej bazie danych. W tym celu została użyta baza SQLite3, oferująca możliwość szybkiego wdrożenia bez konieczności złożonej konfiguracji systemu. W razie potrzeby przechowywania danych wielu klientów istnieje możliwość zamiany silnika bazy danych na bardziej zoptymalizowany pod kątem przechowywania wielu rekordów oraz obsługujący większą ilość zapytań w jednym czasie.



Rys. 5.1. Diagram ERD aplikacji

System w formie bazowej używa tylko jednej tabeli – *users*, przechowującej:

- id – unikalny identyfikator rekordu
- username – nazwa użytkownika, używana do logowania
- passwordPath – ścieżka do pliku zawierającego formułę *NDB* kodującą skrót hasła
- salt – „sól” – parametr algorytmu uzyskiwania klucza na podstawie frazy hasłowej

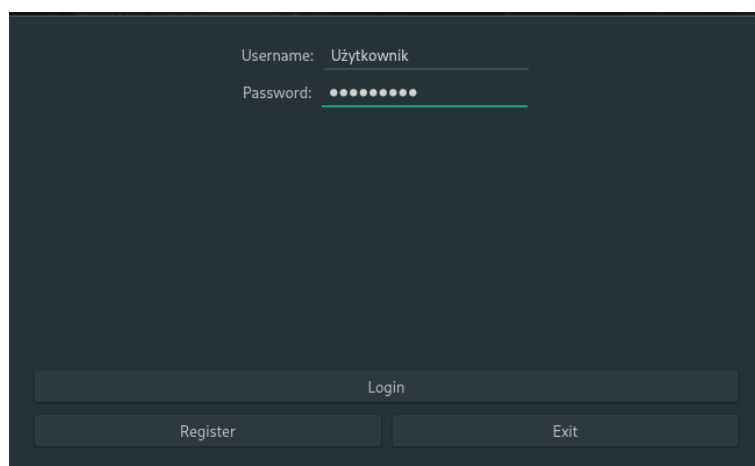
W przypadku potrzeby przechowywania dodatkowych danych należy dodać do tej tabeli klucze obce łączące użytkowników z pozostałymi obiektami pozytywnej bazy danych.

W trakcie procedury tworzenia użytkownika generowany jest skrót hasła za pomocą algorytmu PBKDF2 z użyciem funkcji skrótu SHA-512 oraz losowej soli. Metoda jest ustawiona na 10000 iteracji i generuje skrót o długości 512 bitów. Do generacji *NDB* używany jest algorytm *K-Hidden*, który został wyłoniony na podstawie testów jako najbardziej bezpieczny. Ponieważ jest to niekompletna metoda generacji konieczne jest zawarcie sumy kontrolnej – do tego ponownie używana jest funkcja skrótu SHA-512. Finalny rekord pozytywny jest konkatencją wyniku funkcji PBKDF2 oraz sumy kontrolnej i jego długość wynosi 1024 bity.

Tak skonstruowany ciąg bitowy jest kodowany do postaci negatywnej wykorzystując parametry przy których czas rozwiązywania okazał się najdłuższy tj. $p = \{0.7, 0, 0.3\}$, $k = 3$ i $r = 4.5$. Następnie powstała *NDB* jest zapisywana do pliku w formacie *ndb* (zbiór napisów na alfabecie $\{0, 1, *\}$), którego ścieżka jest zapisana w kolumnie *passwordPath*.

5.2. Działanie aplikacji

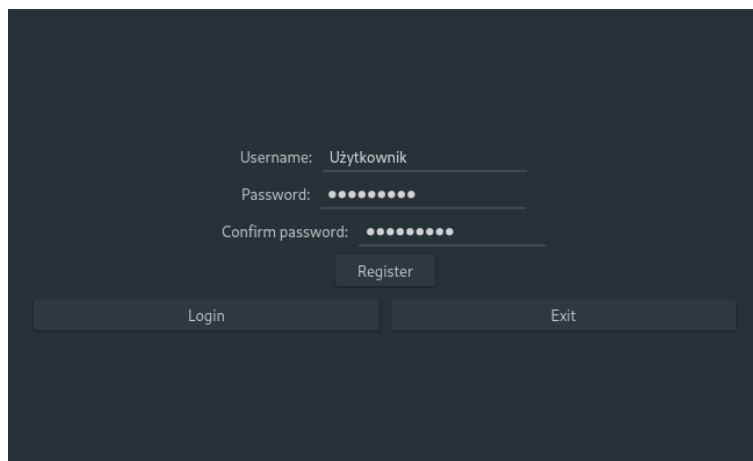
Aplikacja posiada prosty graficzny interfejs użytkownika składający się z trzech widoków - logowania, rejestracji oraz panelu użytkownika. Po uruchomieniu widoczne jest okno logowania. Z tego poziomu można wpisać login i hasło – jeśli system uzna je za poprawne, to następuje przekierowanie do widoku użytkownika.



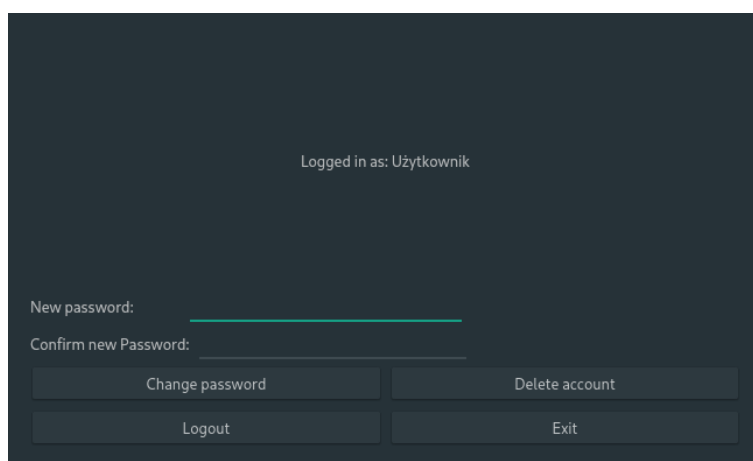
Rys. 5.2. Panel logowania

Po kliknięciu przycisku *Register* na panelu logowania pojawia się widok rejestracji. Tutaj można wprowadzić nowe dane uwierzytelniania i potwierdzić je ponownie naciskając przycisk *Register* – jeśli hasło spełnia wymagania, następuje powrót do widoku loginu, gdzie można uwierzytelnić się nowymi danymi.

Widok użytkownika oferuje podstawowe funkcje manipulacji kontem. Opcja *Change password* udostępnia możliwość zmiany hasła na nowe oraz *Delete account* powoduje usunięcie wszelkich informacji

**Rys. 5.3.** Panel rejestracji

o aktywnym użytkowniku. Po kliknięciu przycisku *Logout* następuje zakończenie sesji i umożliwia ponowne zalogowanie się na inne konto.

**Rys. 5.4.** Panel użytkownika

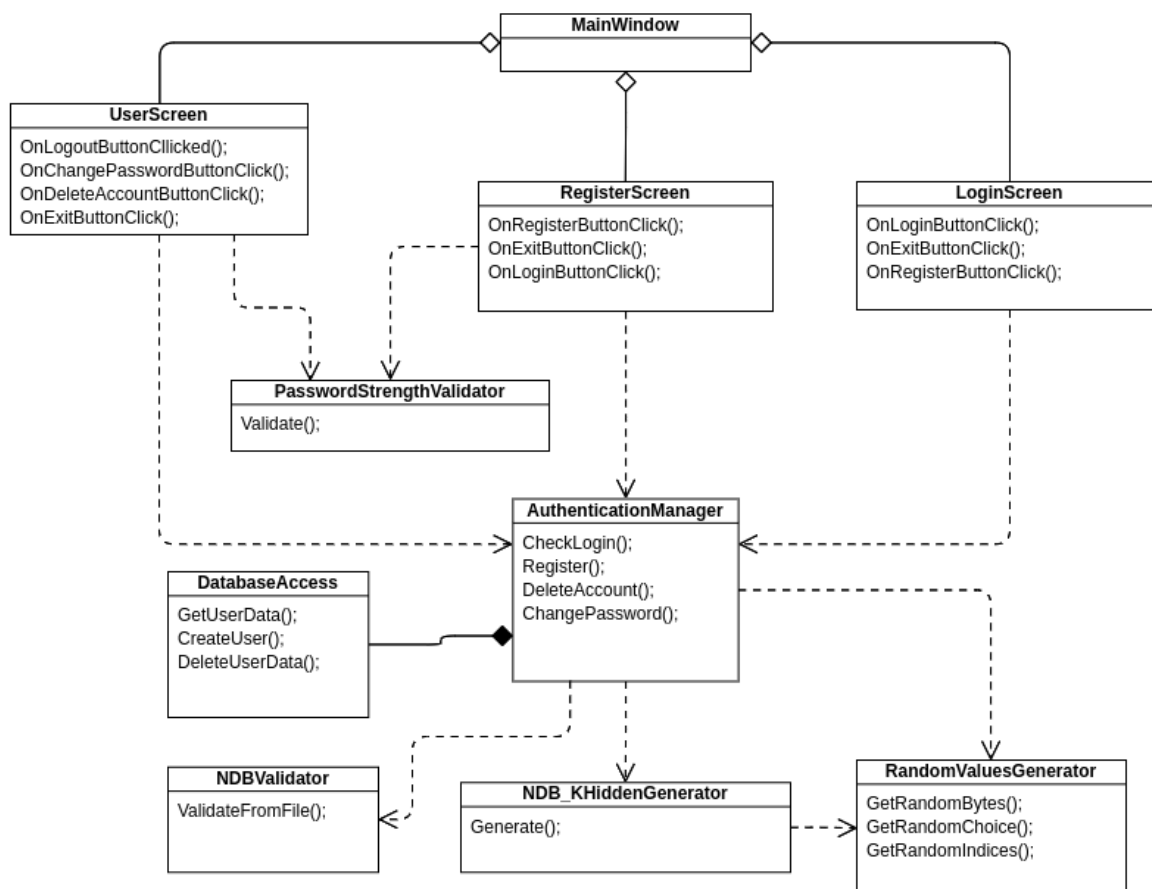
5.3. Szczegóły implementacji

5.3.1. Architektura systemu

Za logikę uwierzytelniania odpowiedzialna jest klasa *AuthenticationManager* oferująca metody do sprawdzenia poprawności danych, tworzenia i usuwania kont oraz zmiany hasła. Wewnętrznie komunikuje się z klasą *DatabaseAccess* będącą warstwą dostępu do użytej pozytywnej bazy danych. Obiekty opisujące poszczególne widoki tj. *LoginScreen*, *RegisterScreen* oraz *UserScreen* są w hierarchii poniżej klasy okna głównego – *MainWindow*. Zbierają dane przekazane przez użytkownika z odpowiednich pól interfejsu i tworzą odpowiednie zapytania do *AuthenticationManager*. Przed oddelegowaniem do procedur tworzenia konta i zmiany hasła dane muszą być zaklasyfikowane jako poprawne za pomocą klasy

PasswordStrengthValidator, sprawdzającej czy dane hasło spełnia zaprogramowane zasady bezpieczeństwa. Żeby tworzenie hasła powiodło się musi mieć następujące cechy:

- Długość musi wynosić 8 lub więcej znaków
- Musi zawierać co najmniej jedną małą literę, jedną wielką literę, jedną cyfrę oraz jeden znak specjalny
- Nie może być identyczne jak nazwa użytkownika (z pominięciem wielkości liter)

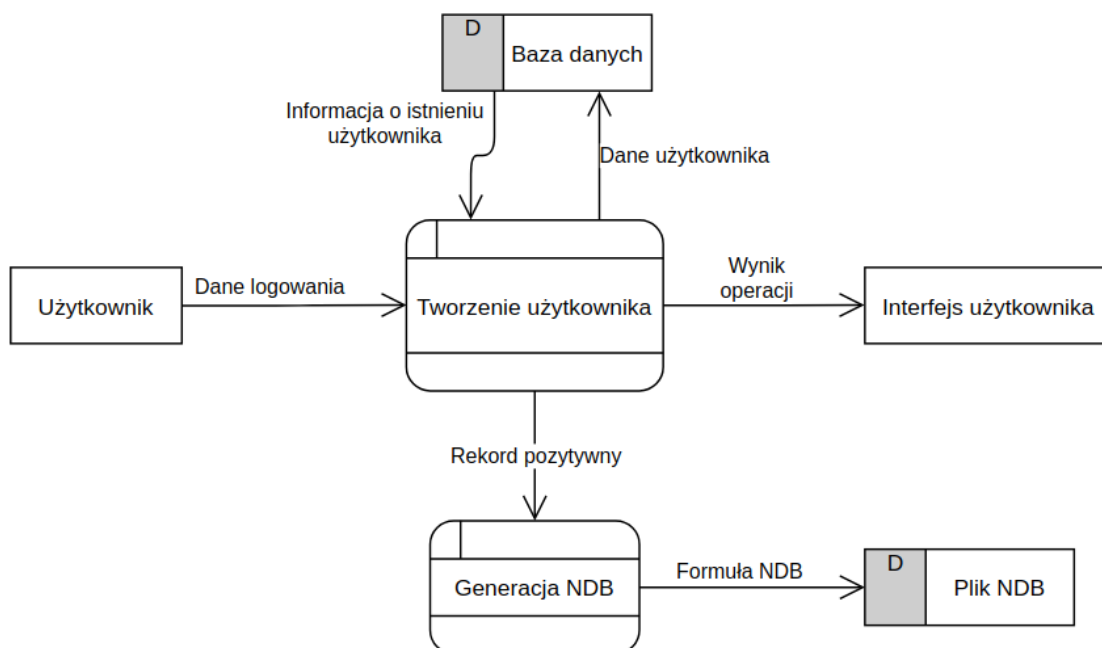


Rys. 5.5. Diagram klas aplikacji

5.3.2. Algorytm tworzenia użytkownika

Jeśli dane zostały uznane jako bezpieczne z punktu widzenia złożoności hasła, obiekty opisujące interfejs tworzą zapytanie o stworzenie konta. Następnie *AuthenticationManager* sprawdza, czy w bazie nie istnieje użytkownik o takiej samej nazwie – jeśli tak, to proces kończy się porażką i użytkownik jest o tym informowany. W innym przypadku tworzony jest rekord *DB*. Sól jest generowana za pomocą klasy pomocniczej *RandomValuesGenerator*. Następnie tworzony jest obiekt *NDB_KHiddenGenerator*

i inicjalizowany odpowiednimi parametrami. Powstała *NDB* jest generowana do pliku i jej ścieżka oraz pozostałe dane umieszczane są w bazie relacyjnej.



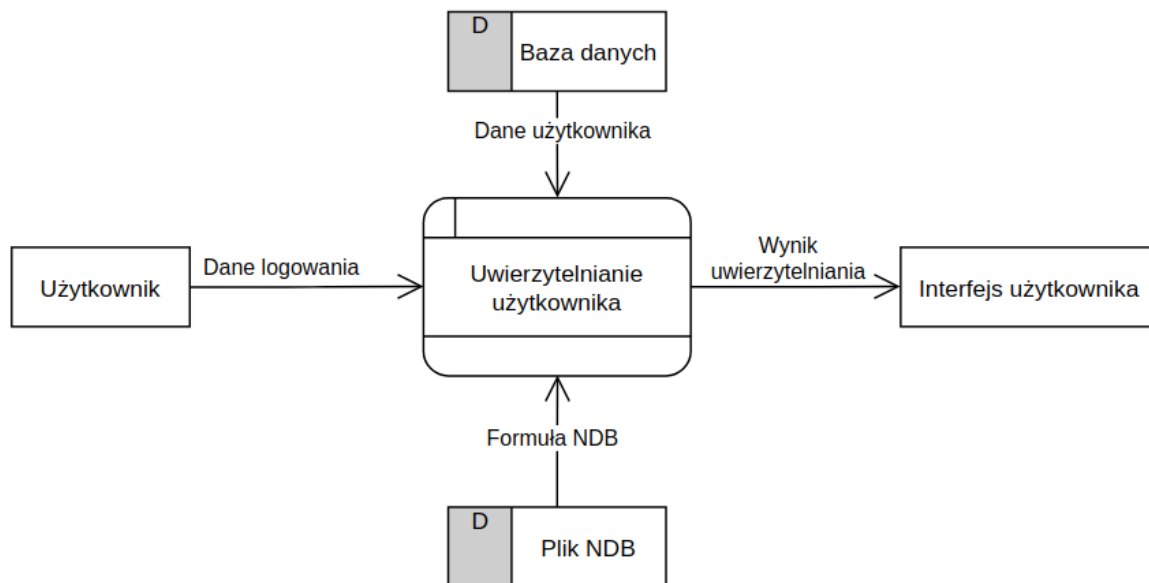
Rys. 5.6. Diagram przepływu danych w algorytmie tworzenia użytkownika

5.3.3. Algorytm uwierzytelniania użytkownika

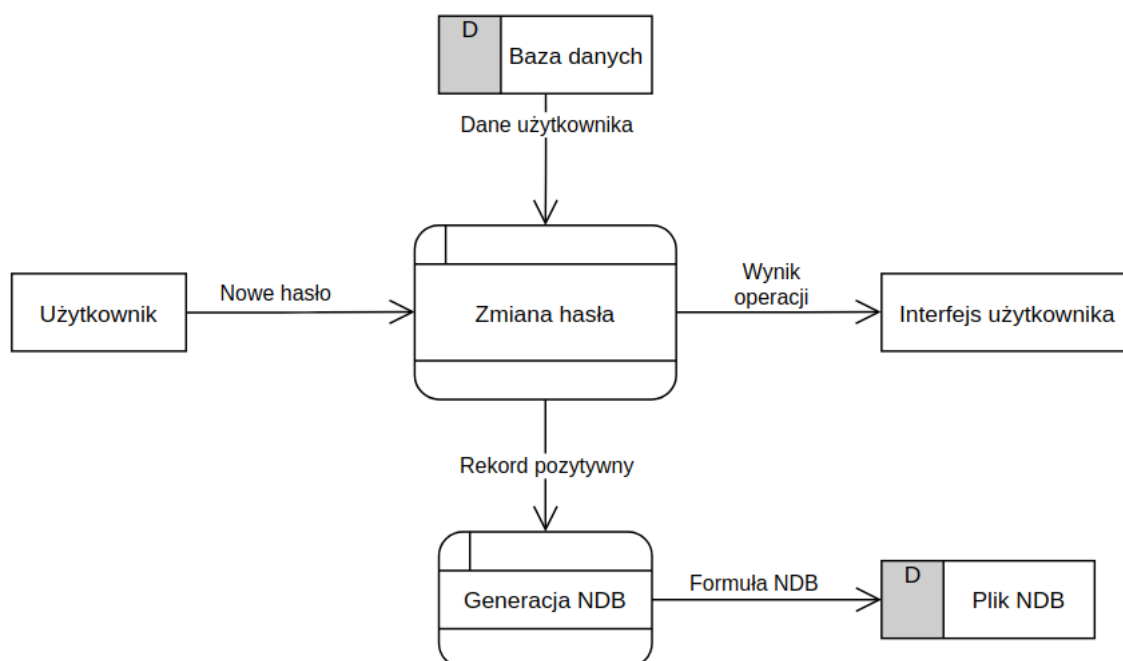
Po uzyskaniu od użytkownika loginu i hasła dane konta są pobierane z bazy SQLite3, jeżeli istnieją. Następnie wczytywany jest plik *NDB* za pomocą klasy *NDBValidator*, która sprawdza czy rekord zawierający skrót danego hasła faktycznie nie pokrywa się z żadnym rekordem negatywnym. W takim przypadku *AuthenticationManager* zwraca sygnał, że użytkownik powinien mieć dostęp do danych.

5.3.4. Algorytm zmiany hasła

Po otrzymaniu zapytania o zmianę hasła pobierana jest ścieżka do pliku dotychczasowej *NDB* i generowany jest nowy rekord *DB* na podstawie algorytmu tworzenia użytkownika. Następnie inicjalizowany jest nowy obiekt generatora *NDB* i wynik jego działania zastępuje dotychczasowe hasło w postaci negatywnej.



Rys. 5.7. Diagram przepływu danych w algorytmie logowania



Rys. 5.8. Diagram przepływu danych w algorytmie zmiany hasła

6. Wnioski

6.1. Bezpieczeństwo przedstawionej implementacji

Z testów przedstawionych w rozdziale 4 wynika, że istnieją algorytmy *NDB* pozwalające na generowanie trudnych formuł SAT, które są rozwiązywalne w czasie wykładniczym. Zostało to wykorzystane podczas implementacji aplikacji umożliwiającej uwierzytelnianie użytkowników za pomocą loginu i hasła. Użycie *NDB* zamiast pozytywnej bazy danych do składowania skrótu hasła nie zwiększa drastycznie bezpieczeństwa w stosunku do dobrze zaprojektowanego systemu, używającego odpowiednich procedur kryptograficznych, w szczególności algorytmu uzyskiwania klucza na podstawie hasła. Jednak obecność dodatkowej warstwy może odeprzeć atak na tak zaprojektowaną aplikację, gdy pozostałe metody zawiodą.

Sam fakt użycia *NDB* nie rozwiąże głównych problemów z systemami uwierzytelniania, zwłaszcza stosowania słabych i przewidywalnych haseł, dlatego w implementacji aplikacji wymuszam wybór fraz o zwiększonej złożoności. Dodatkowo konieczność porównania klucza z rekordami negatywnym może spowolnić próby złamania przez metody *brute-force*.

Negatywne bazy danych mogą oferować znacznie większą poprawę bezpieczeństwa w systemach, w których nie jest konieczne powiązanie rekordów z dodatkowymi danymi. W tym celu należy zrezygnować z większości zalet posiadanych przez struktury przechowywania danych w postaci pozytywnej i zmodyfikować problem do takiego, który wykorzystuje tylko jedną operację – sprawdzenie obecności danego rekordu w bazie.

6.2. Możliwe rozszerzenia

Przedstawiona aplikacja, poza dodaniem referencji do dodatkowych danych opisujących użytkownika może być dowolnie zmodyfikowana w celu spełnienia określonych wymagań biznesowych.

Jedną z opcji dalszego zwiększenia bezpieczeństwa może być zaimplementowanie uwierzytelniania dwuskładnikowego, uniemożliwiającego dostęp do konta w przypadku uzyskania dostępu do tylko jednego składnika. Żeby zapewnić stosowanie dobrych nawyków podczas korzystania z aplikacji możliwe jest także stworzenie modułu wymuszającego częste zmiany hasła.

Bibliografia

- [1] Google / Harris Poll. *The United States of P@ssw0rd\$*. <https://storage.googleapis.com/gweb-uniblog-publish-prod/documents/PasswordCheckup-HarrisPoll-InfographicFINAL.pdf>. 2019.
- [2] W. Luo i in. „Three Branches of Negative Representation of Information: A Survey”. W: *IEEE Transactions on Emerging Topics in Computational Intelligence* 2.6 (2018), s. 411–425. DOI: 10.1109/TETCI.2018.2829907.
- [3] Stephen A. Cook. „The Complexity of Theorem-Proving Procedures”. W: *Proceedings of the Third Annual ACM Symposium on Theory of Computing*. STOC '71. Shaker Heights, Ohio, USA: Association for Computing Machinery, 1971, 151–158. ISBN: 9781450374644. DOI: 10.1145/800157.805047.
- [4] B. A. Trakhtenbrot. „A Survey of Russian Approaches to Perebor (Brute-Force Searches) Algorithms”. W: *Annals of the History of Computing* 6.4 (1984), s. 384–400. DOI: 10.1109/MAHC.1984.10036.
- [5] Thomas J. Schaefer. „The Complexity of Satisfiability Problems”. W: *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*. STOC '78. San Diego, California, USA: Association for Computing Machinery, 1978, 216–226. ISBN: 9781450374378. DOI: 10.1145/800133.804350.
- [6] Adnan Darwiche i Knot Pipatsrisawat. „Complete Algorithms”. W: *Handbook of Satisfiability*. Red. Armin Biere i in. T. 185. Frontiers in Artificial Intelligence and Applications. IOS Press, 2009, s. 99–130. DOI: 10.3233/978-1-58603-929-5-99.
- [7] Henry A. Kautz, Ashish Sabharwal i Bart Selman. „Incomplete Algorithms”. W: *Handbook of Satisfiability*. Red. Armin Biere i in. T. 185. Frontiers in Artificial Intelligence and Applications. IOS Press, 2009, s. 185–203. DOI: 10.3233/978-1-58603-929-5-185.
- [8] Fernando Esponda. “Negative Representations of Information”. PhD thesis. 2005.
- [9] F. Esponda, S. Forrest i P. Helman. „Enhancing Privacy through Negative Representations of Data”. W: 2004.
- [10] Haixia Jia, Cristopher Moore i Doug Strain. „Generating Hard Satisfiable Formulas by Hiding Solutions Deceptively”. W: *Journal of Artificial Intelligence Research - JAIR* 28 (mar. 2005). DOI: 10.1613/jair.2039.

- [11] Dimitris Achlioptas, Haixia Jia i Christopher Moore. „Hiding Satisfying Assignments: Two are Better than One”. W: *Proceedings of the National Conference on Artificial Intelligence* 24 (kw. 2005). DOI: [10.1613/jair.1681](https://doi.org/10.1613/jair.1681).
- [12] Fernando Esponda i in. „Protecting Data Privacy Through Hard-to-Reverse Negative Databases”. W: *Information Security*. Red. Sokratis K. Katsikas i in. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006. ISBN: 978-3-540-38343-7.
- [13] D. Zhao i in. „A fine-grained algorithm for generating hard-to-reverse negative databases”. W: *2015 International Workshop on Artificial Immune Systems (AIS)*. 2015, s. 1–8. DOI: [10.1109/AISW.2015.7469244](https://doi.org/10.1109/AISW.2015.7469244).
- [14] R. Liu, W. Luo i L. Yue. „The p-hidden algorithm: Hiding single databases more deeply”. W: *Immune Computation* 2 (sty. 2014), s. 43–55.
- [15] R. Liu, W. Luo i X. Wang. „A Hybrid of the prefix algorithm and the q-hidden algorithm for generating single negative databases”. W: *2011 IEEE Symposium on Computational Intelligence in Cyber Security (CICS)*. 2011, s. 31–38. DOI: [10.1109/CICYBS.2011.5949400](https://doi.org/10.1109/CICYBS.2011.5949400).
- [16] Bart Selman i Henry Kautz. *Stochastic Local Search for Satisfiability*. <https://www.cs.rochester.edu/u/kautz/walksat/>.
- [17] Princeton University. *zChaff*. <https://www.princeton.edu/~chaff/zchaff.html>.